Hai Cu The

Secure Web Service for a Windows Phone Application

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

9 April 2016

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Hai Cu The Secure Web Service for a Windows Phone Application 49 pages + 1 appendix 9 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Instructor(s)	Petri Vesikivi, Principal Lecturer

The purpose of this study is to learn the concepts, algorithms and techniques of cryptography and how to apply them to the web service related to Windows Phone applications. The method is to study the mechanism of the cryptographic algorithms and selecting the suitable techniques for Windows Phone – Azure in a Windows ecosystem. The best practices and popular algorithms are investigated in detail to offer the benefits and drawbacks when applying them to the ecosystem.

In the case study, a proof-of-concept server-client Windows Phone application is implemented. Both the server and client use the mentioned theories to build a secure communication channel between the client and server. It applies both symmetric key and asymmetric key cryptography for encrypting and decrypting the data and also uses client certification authentication and server certificate to prevent from eavesdroppers and man-in-the-middle attacks.

In conclusion, the study shows the crucial requirements of a secure Windows Phone application and its backend. The developers need to comprehend sufficient knowledge of cryptography, encryption algorithms and application security to be able to develop a secure application and service.

Keywords	Windows Phone, azure, cryptography, certificate, application security

Contents

Abbreviations and Acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certificate Authority
CBC	Cipher Block Chaining
CSR	Certificate Signing Request
CTA	Call to Action
DES	Data Encryption Standard
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IV	Initialization Vector
JSON	JavaScript Object Notation
MACs	Message Authentication Codes
NIST	National Institute of Standards and Technology
OS	Operating System
PFX	Personal Information Exchange
PKP	Pretty Good Privacy
PKI	Public Key Infrastructure
RA	Registration Authority
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer
TLS	Transport Layer Security
WP	Windows Phone

1 Introduction

From the beginning of the information technology era, information security has always been an essential subject. With the increasing number of web services and applications, the amount of confidential data transferring on the internet is also growing tremendously. For that reason, protecting the data from being breached and the web services from being attacked is considered much more important than ever. The data must be encrypted from the sender before sending it to the recipient. Moreover, the communication channel between the parties needs to be protected from unauthorized access and attackers.

This thesis is dedicated to study the theories, concepts and applications related to cryptography, a technique to transfer data from a meaningful form to an unintelligible form. The best practices and most well-known algorithms are also studied and applied to improve the security of web services and mobile applications. The case study is about the proof-of-concept Windows Phone application and its web service using the .NET framework and hosted in an Azure cloud. The web service applies the best practice to create a secure channel to the application. The data transferred between the web service and the mobile application is also encrypted with both a symmetric key and public key algorithms.

Windows Phone was introduced by Microsoft in 2010 as the successor of Windows Mobile to compete with the success of Apple's iOS and Google's Android [1]. In the meantime, Nokia decided to abandon Symbian and partnered with Microsoft and use Windows Phone as their main platform [2]. Windows Phone gained the market share slowly to became the third popular smartphone OS [3].

2 Windows Phone Platform

2.1 History

Microsoft released the first handheld portable platform in 1996 with the very first version of Windows Embedded Compact (Windows CE) [4]. Originally, Windows CE is the operating system for mobile devices as part of the Windows Embedded family of products.

The OS is optimized for low memory and processing speed device. Although Microsoft switched to Windows Mobile and Windows Phone later as the main OS for their smartphone, Windows CE is keep upgrading and released alongside. Currently, the OS is applied mainly for the embedded systems and industrial devices. [5.]

As figure 1 shows, in 2000, Microsoft released the first version of the Pocket PC (2000) operating system. Pocket PC is developed based on Windows CE 3.0 and is able to run the mobile version of Microsoft Office. In October 2001, Pocket PC 2002 was introduced with some mobile phone features in addition to the personal digital assistant (PDA) abilities [6].

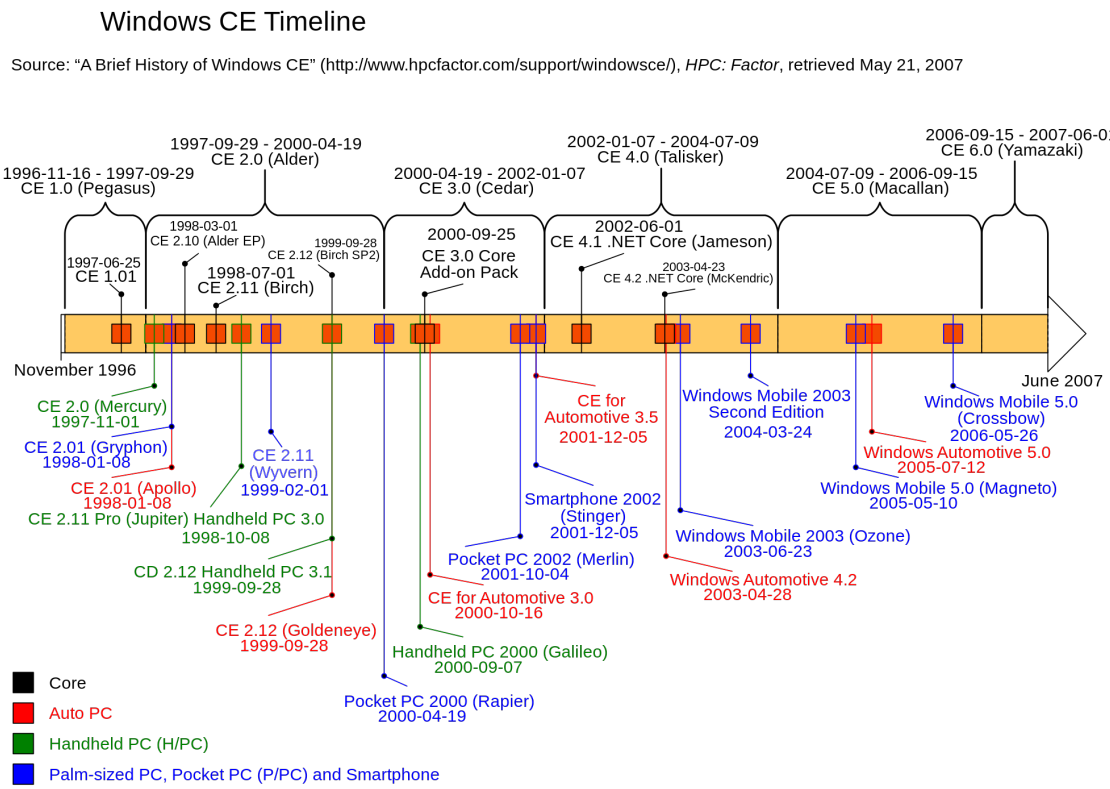

Figure 1. Windows CE timeline. Reprinted from Tele-Task.de [7].

Figure 1 also illustrates that Microsoft released the first functional mobile phone operating system, named "Windows Mobile 2003" which is powered by Windows CE 4.2 in 2003. This version was enhanced with the Bluetooth feature and some built-in applications such as Pocket Outlook, Pocket Internet Explorer and SMS. [8.]

Two years later, Windows Mobile 5.0 was introduced. The OS integrates the .Net Compact Framework 1.0 for the .Net programs such as Microsoft Exchange Server, Microsoft

Office Mobile and Windows Media Player 10 Mobile. [9.] During the next five years, more versions of Windows Mobile were released like Windows Mobile 6, 6.1 or 6.5.

In 2004, Windows Mobile gained 23% of worldwide smartphone sales and quickly increased to 42% in 2007 at its peak. However, it started to fall off steadily due to the competitiveness from iOS and Android. [10.] Meanwhile, Microsoft realized the irresponsiveness and weakness in touch screens of this operating system and planed to replace it with a whole new smartphone operating system. In October 2010, Microsoft officially introduced Windows Phone as the replacement for its predecessor, Windows Mobile. [11.]

2.2 Platform Overview

Since the beginning of the Windows Phone era, Microsoft released four different generations of Windows Phone operating systems in total, which are Windows Phone 7, Windows Phone 8, Windows Phone 8.1 and the newly released Windows 10 Mobile in 2016.

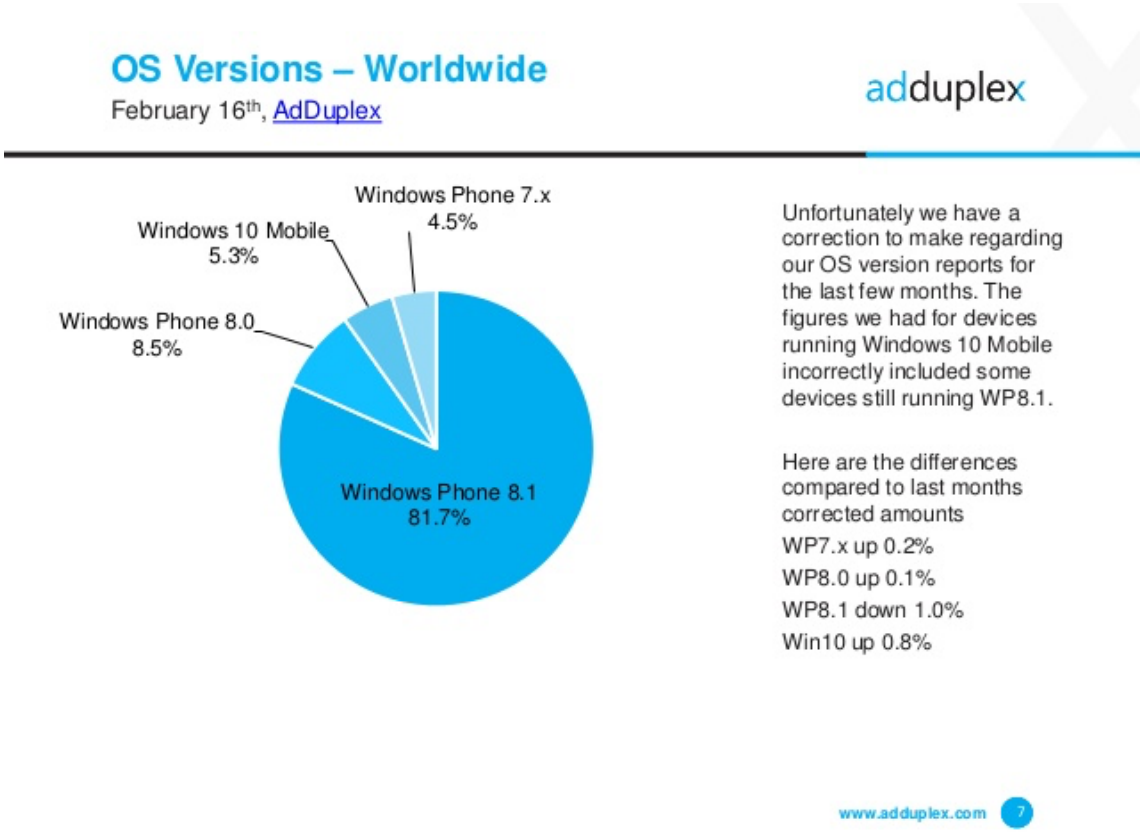

Figure 2. Windows Phone OS Versions - Worldwide, February 2016. Reprinted from AdDuplex. [12]

Figure 2 illustrates the share of the Windows Phone OS versions in February 2016 by Adduplex. Windows Phone 8.1 OS is still ruling with more than 80% of the whole Windows Phone market share but the share of Windows 10 Mobile is also increasing gradually because of Microsoft's effort to converge Windows 10 devices.

From Windows Phone 8 onwards, Microsoft abandoned the Windows CE-based architecture, which is still used in Windows Phone 7, and replaced it with the Windows NT kernel-based one. Many components from Windows 8 were shared in the Windows Phone 8 version, allowing the application to be ported between two platforms. The successor of Windows Phone 8 is Windows Phone 8.1, which was released in July 2014 [13]. Because of the similarity of the hardware requirements, all Windows Phone 8 devices are able to upgrade to Windows Phone 8.1. Below the minimum hardware requirements for Windows Phone 8.1 are listed [14]:

- One of these Qualcomm processor: MSM8x10, MSM8x12, MSM8x26, MSM8916, MSM8926, MSM8x28, MSM8928, MSM8974, MSM8974Pro,

MSM8960, MSM8260A, MSM8660A, MSM8930AA, MSM8930AB, MSM8630, MSM8230, MSM8627, MSM8227, MSM8994, MSM8992, MSM8952, MSM8909, MSM8208.

- At least 512 MB RAM for 800x480 or 854x480 display resolution.
- Multi-touch capacitive touch screen with minimum of four simultaneous points
- Minimum 8 GB flash memory.
- Less than 8-inches screen with 32 bits of color per pixel.
- Hardware buttons such as Power button, Volume Up and Volume Down.
- Proximity sensors, accelerometer, and vibration motor.
- Micro-USB 2.0 support
- 802.11b/g and Bluetooth (802.11n is optional)
- (magnetometer, gyroscope and ambient light sensors are optional)
- Rear-facing AF camera with optional LED or Xenon flash, optional front-facing camera (both need to be VGA or better)

In Windows Phone 8.1, many new features were included such as the notification center, volume control separation or the voice assistant Cortana.

In early 2015, Microsoft announced the new mobile operating system for smartphones and tablets running on ARM architectures as Windows 10 Mobile as the counterpart of Windows 10 for PC [15]. Microsoft gets rid of “Windows Phone” as the OS name due to branding this OS as an edition of Windows 10. However, the hardware of Windows Phone 8.1 devices can still be upgraded to Windows 10 Mobile. The main focus for Windows 10 Mobile is to provide consistency in terms of user experiences and functionality with its counterpart for PC. Under the Universal Windows Platform concept, the Windows 10 family runs on the same codebase, and therefore, the application for Windows 10 is able to run on Windows 10 Mobile and vice verse. Moreover, a new feature, called “Continuum”, is also supported in the new Windows 10 Mobile devices. Continuum allows the smartphone to run as a PC-like desktop by connecting it to an external display. The device can either connect to the external display via Miracast, USB Type-C or via a docking station, called “Display Dock”, with the USB ports for a mouse, keyboard and HDMI output. [16.] Furthermore, Microsoft announced the middleware tools for porting the Android and iOS application to Windows 10 device for attempting to overcome the lack of Windows Phone applications in the Windows Phone store [17].

The third party such as indie developers or companies can develop the Windows Phone applications by using C#/Visual Basic.NET, C++ or HTML5/JavaScript. Microsoft also provides many different software and tools to help the developer to design, develop and also test the application more easily such as like Microsoft Visual Studio IDE, Windows Phone Developer Tools, Express Blend for Windows Phone or Windows Phone Emulators. Games for Windows Phone can be developed based on XNA platforms or some other game engines such as Unity or Unreal Engine (for Windows 10 Mobile). Afterwards, the applications and games need to be submitted to the Store via the App Hub for verifications and validations from Microsoft to check if they meet the standardization criteria set. The price of the application or the game is decided by the developer but 20% of the revenue is paid to Microsoft.

The scope of this thesis only covers the snippet code for Windows Phone 8.1 application in the Win RT framework.

2.3 Architecture

Figure 3 explains the high-level architecture of Windows Phone and the association between Microsoft and their Original Equipment Manufacturers (OEMs). Boot drivers for the phone and most of the low-level hardware interaction are provided by partners in the form of a board support package (BSP). The suppliers of CPU for the phone, also known as the Silicon Vendors (SV), might write the core of the BSP. However, OEMS are still responsible for delivering all required drivers for the phone hardware. Meanwhile, Microsoft provides the kernel and Windows Phone OS. The top layers, above the kernel layer in figure 3, contains the system services such as Shell services, DirectX and Radio telephony and the programming frameworks that run the application of the phone. [18.]

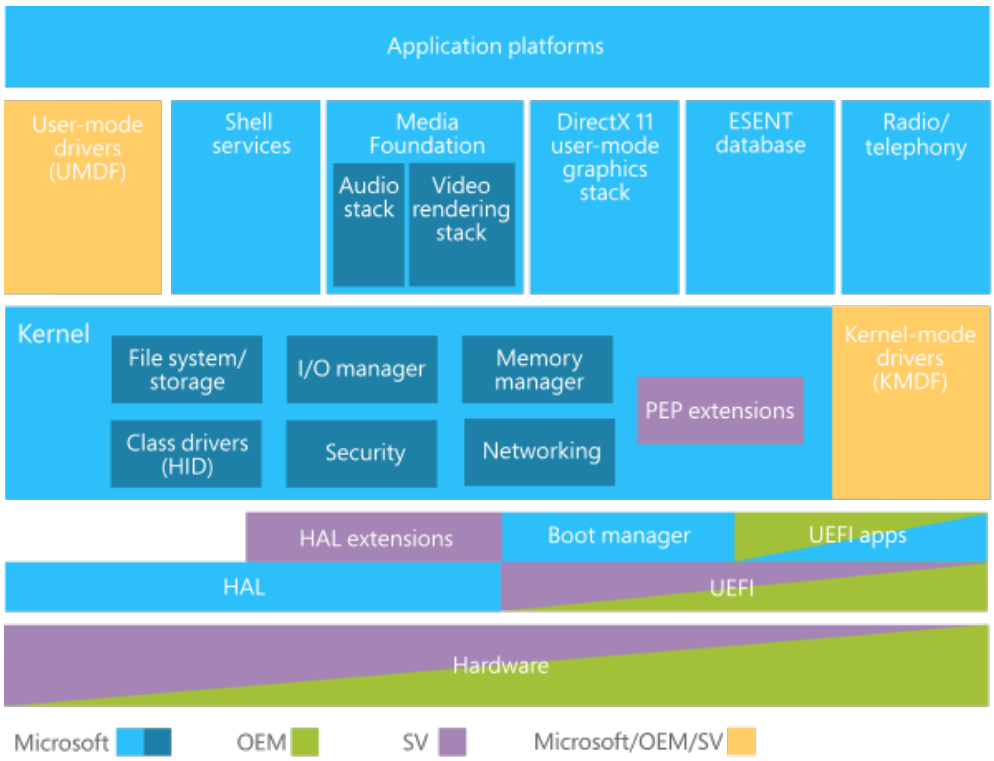

Figure 3. Windows Phone architecture overview. Reprinted from Microsoft.com. [18]

The application developers do not need to care about the low-level APIs or services. Their main work is to develop the application based on the API from the application platforms.

2.4 Security for Windows Phone Application Development

Security is one of the most important parts for the application development, especially when the application is the client side for web services. All the user's information and other data needs to be protected as carefully as possible. For example, all the data or messages between the client side and the server side should be encrypted and run on a secure channel to prevent them from breach. Therefore, Microsoft designed many security modules for Windows Phone application development to help users to have the best and comfortable phone experience at the same time.

2.4.1 Third-party Applications in the Windows Phone Store

Similar to iOS and Android, the Windows Phone platform has its own application store to let the user to download and use the third-party application on the device. To ensure the quality of user experience and that the user's personal information will not be leaked or accessed without permission, the application developers need to take extra care as regards the security and performance of the application. They need to integrate the security safeguards into the development environment and lifecycle. Moreover, the development devices also need to be registered as testing devices to be able to run or debug the application. Before submitting the app to the store, it needs to pass all the publicly documented certification tests. If the app passes all the tests, it will be digitally signed on behalf of the developer and be ready to publish on the Windows Phone Store. [19.]

2.4.2 Application Safeguards

The Windows Phone app platform uses a various number of technologies to protect users from applications that include suspicious and unexpected behavior [19]:

- Windows Phone applications are run in a sandboxed process and isolated from each other. They can only interact with the phone features and with each other through a strictly controlled mechanism.
- The sandboxed process is also designed to have a customized set of security capabilities to reduce the attack surface area of the application.
- The Windows Phone Store is the only source that provides the legitimate application acquisition and application licenses with mandatory code signing. It helps to set the standards for the Windows Phone application.
- An execution manager supervises and measures the Windows Phone applications if the applications' behaviors are following a certain defined standard.

2.4.3 Data Encryption and Supported Cryptographic Algorithms

Microsoft also provides many solutions to encrypt and decrypt confidential data such as passwords, connection strings, and PINs in a Windows Phone app by using Data Protection API (DPAPI). Besides, Windows Phone also supports many cryptographic algorithms such as AES, HMACSHA1, HMACSHA256, Rfc2898DeriveBytes, RSA, SHA1, SHA256. [19.]

3 Cryptography

Cryptography is the technique for information hiding and verification by using the algorithms, protocols and strategies. The purpose of cryptography is to secure communication between two parties and prevent an unauthorized access from reading sensitive information. Cryptography is applied widely in many aspects from secure network communication to physical security mechanisms such as smartcards or biometrics.

The process of transforming information from understandable form (termed **plaintext**) to unintelligible form (termed **ciphertext**) is called **encryption**. A reverse process is named as **decryption**. The method or the algorithm used to transform the plaintext into ciphertext and vise versa is called a **cipher**. The knowledge of the cipher is sharable; therefore in practice, the cipher encrypts or decrypts a message with a **cryptographic key**. The key is usually a string of character and sharable within the intended recipients or authorized parties.

Cryptography is one of the techniques in information security, and therefore it also needs to possess the basic principles of information security. There are four main principles for cryptography [20,4]:

- Confidentiality (Privacy): The information is only available to authorized individuals, parties and only they can decrypt the content of the message.
- Integrity: The information has to keep its accuracy and completeness over the encryption and decryption processes.
- Authentication: The identity of the sender or the origin should be verifiable from the recipients.
- Non-repudiation: The senders are obligated to send the message. They cannot refuse the task.

Based on how the key is used, cryptography can be divided into two different cryptosystems: symmetric key cryptography and asymmetric key cryptography, also known as “public key cryptography”.

3.1 Symmetric Key Cryptography

Symmetric-key cryptography, also called private-key cryptography, is the technique using the same cryptographic key for both encryption and decryption. Both the sender and the recipient need to have access to the cryptographic key to encrypt or decrypt the message as illustrated in figure 4. It is the shared secret key between them.

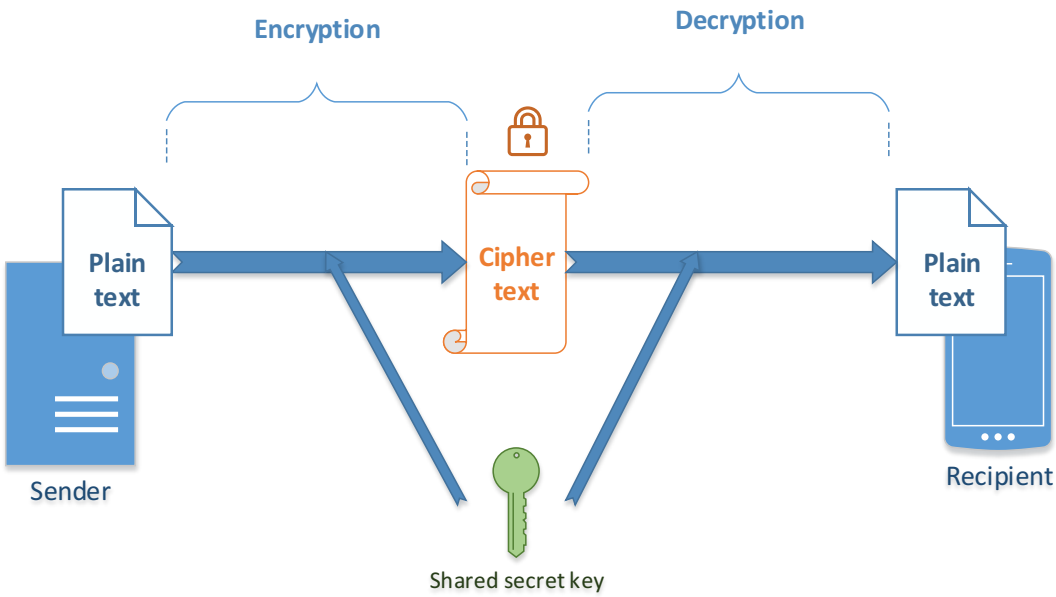

Figure 4. Symmetric key cryptography scheme.

In practice, asymmetric cryptography can be applied to share the key between the parties via the internet. Symmetric key ciphers are separated into either **block ciphers** or **stream ciphers**.

The stream cipher enciphers every single digit from the plaintext stream to the corresponding digit of the ciphertext steam by combining them with an infinity pseudorandom keystream [20, 192-194]. Practically, the combining operation is an exclusive-or (XOR) and one digit is equivalent to a bit. By using digital shift registers, the random seed value generates the pseudorandom. The seed value is also acted as the cryptographic key for encryption and decryption. Generally, the performance of the stream cipher is better than the block cipher. However, it also faces some serious security issues if it is not applied

correctly. Some popular stream ciphers are, for example, RC4, A5/1, A5/2, FISH and Helix.

The block cipher encrypts a fix-length number of bits (a block) from the plaintext to the ciphertext block with the cryptographic key [20, 224-225]. The block cipher can act like the stream cipher depending on the mode of operation is used. The mode of operation explains how to use the cipher's single-block operation repeatedly to convert the data larger than a block. There are several popular modes of operation for the block cipher [20,228]:

- Cipher Block Chaining(CBC)
- Electronic Codebook (ECB)
- Output Feedback (OFB)
- Cipher Feedback (CFB).

Many modes of operation require an initialization vector (IV) for each encryption operation. The IV is a unique binary sequence which must be random and nonrepeating. By using the IV, the same plaintext can be encrypted into different ciphertexts even with the same cryptographic key. Reusing the IV is not recommended because it might leak the information from the plaintext or worse. [20,229.] Another factor that affects the block cipher is padding. The block cipher needs to have fix-length blocks to encrypt but the length of the plaintext varies. Therefore, the last block is usually padded before encryption. The simplest way is to keep adding the null bytes to the last block until its length has been matched. [20, 335.]

Some popular block ciphers are DES, Triple DES, Rijndael / AES, IDEA, Blowfish, Twofish. Among them, the Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are the ciphers standardized by the US government.

3.1.1 Advanced Encryption Standard

Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the U.S. government in 2001 and is used worldwide nowadays. Initially, Rijndael was a family of ciphers developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted it during the AES selection process. After the process, only three members of the Rijndael family were selected. They had a 128-bit block size with three different key lengths 128, 192 and 256

bits [21, 5-7]. From 2009, AES has been one of the most popular algorithms used in symmetric key cryptography.

AES is a substitution-permutation network cipher, which combines both substitution and permutation into a series of linked mathematical operations. A 128-bit block can be interpreted into 16 bytes and operated as a 4x4 column-major order, called “state”.

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

The key size determines the number of repetitions of transformation rounds in order to convert plaintext into ciphertext. It requires 10 cycles of repetition for 128-bit keys while 12 cycles and 14 cycles are for 192-bit keys and 256-bit keys. The transformation round is demonstrated in figure 5.

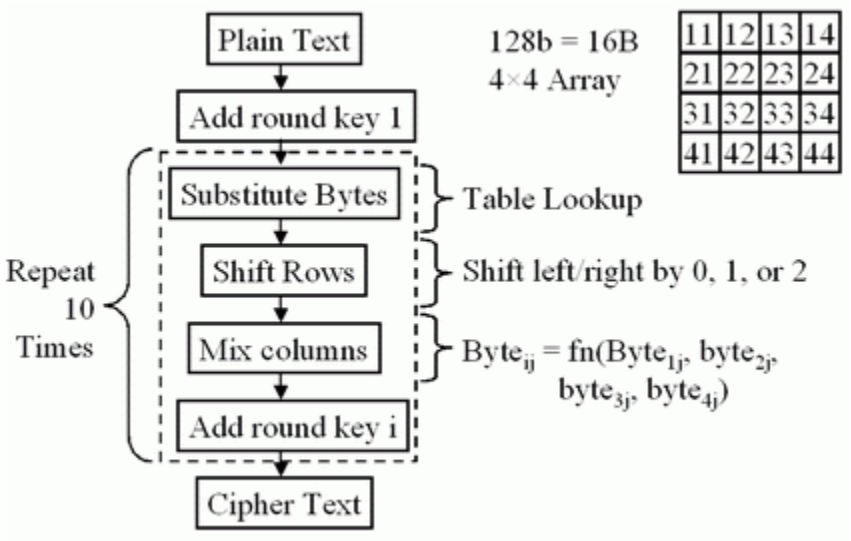

Figure 5. High-level flow diagram of AES [22, 13-14].

Before starting the transformation round, each block of the cryptographic key is combined with the corresponding byte of the state using bitwise XOR, also known as “Add round key” step. Inside the round, it follows the steps: Substitute Bytes, Shift Rows, Mix Columns and Add Round Key. The final round skips the Mix Column step. The output of the final round is the cipher text. [21, 14.]

Until 2009, side-channel attacks were the only successful attacks against the AES. Side-channel attacks are not the attacks to the underlying cipher but to some certain implementations of AES. The US Government still considers the AES secure enough for non-classified data.

3.1.2 Cryptographic Hash Function and Secure Hash Algorithm

A **cryptographic hash function** is the one-way function that produces a fixed-length hash value from the data of an arbitrary size. The hash value result is called the message digest or the **digest**. The cryptographic hash function can compute the digest quickly for any original message and is very sensitive with any changes from the message. If the message is modified, the digest will also be changed. Therefore, it is usually the way to authenticate the content of the message. The recipient can check the message's integrity by comparing the digest attached along with the message to the actual digest calculated from the message itself. Moreover, it is impossible to recreate the input data from the digest alone. Figure 6 shows some example inputs and their digest computed by the hash function.

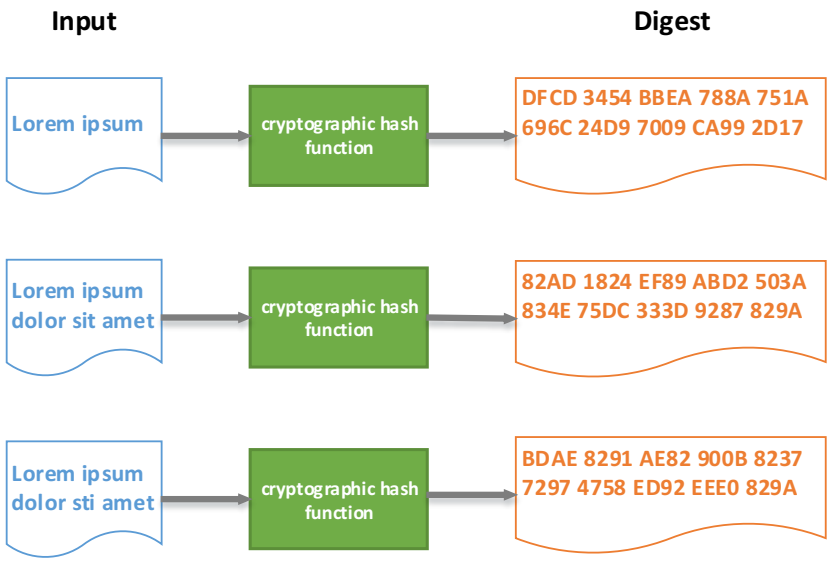

Figure 6. Cryptographic hash function examples.

Cryptographic hash functions have been involved in many information security applications such as digital signatures, message authentication codes (MACs) and calculating the checksums or digital fingerprints [23, 186].

The Secure Hash Algorithm (SHA) is a family of cryptographic hash functions based on block ciphers published by the National Institute of Standards and Technology (NIST). SHA-0 and SHA-1 are deprecated due to security vulnerabilities. Currently, the SHA-2 hash function family and the SHA-3 hash function are the most secure ones for hashing. SHA-2 contains hash functions with the largest hash being 512 bits. [23, 168.]

3.2 Public Key Cryptography

Public key cryptography, also known as asymmetric key cryptography, is the technique for encryption with a pair of keys, the public key and private key. The **public key** is used for encryption while the **private key** is for decryption. Although the public key and private key are related mathematically, it is infeasible to calculate the private key from the public key. Therefore, the public key is published without security concerns, while the private key needs to be kept private. Comparing to the symmetric key algorithm, the public key one is more complex and slower. Thus it is usually applied to encrypt some short messages like the symmetric encryption key. Figure 7 shows the relationship between the public key and private key.

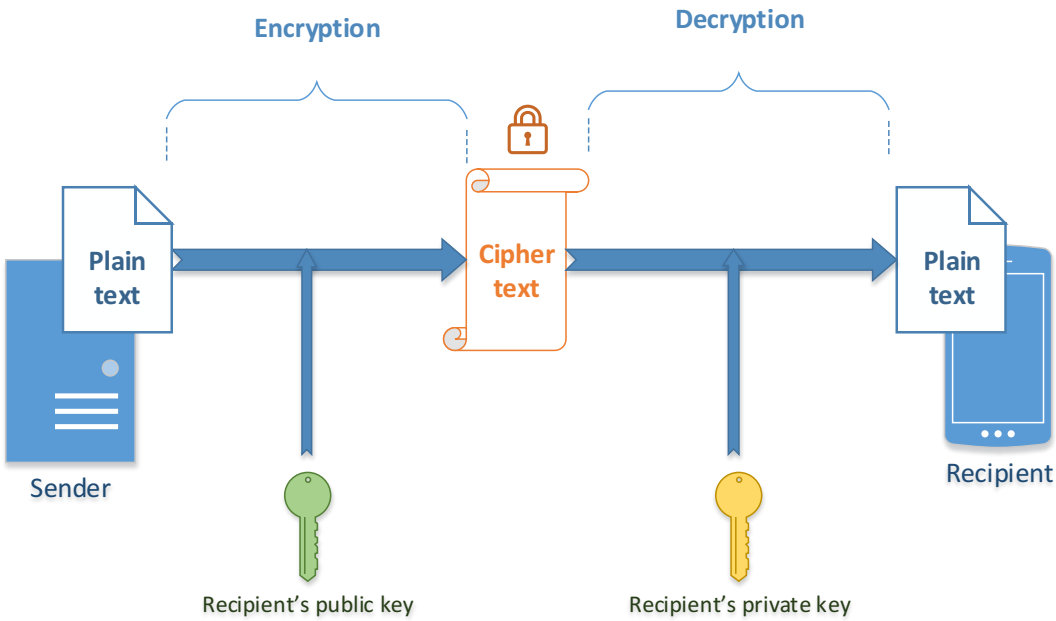

Figure 7. Public key encryption scheme.

There are two best-known uses of public key cryptography:

- **Digital signature**: The purpose is to certify the non-repudiation of the message being sent. The message is signed with the sender's private key, then sent to the recipient. The recipient can verify if the message is really sent from the sender by using the sender's public key.

- **Public-key encryption**: The purpose is to certify communication between the parties is kept confidential during transits. The message is encrypted with the recipient's public key, and sent to the recipient. Only the recipient with his private key can decrypt the encrypted message from the sender. Figure 7 describes one example of public-key encryption.

3.2.1 RSA Algorithm

RSA is a public-key algorithm applied widely for secure data transmission. The algorithm was described in 1977 by Ron Rivest, Adi Shamir and Len Adleman from MIT (Massachusetts Institute of Technology). The name RSA comes from the initials of their surnames. Based on the concept of public-key cryptography, the RSA contains both the public key for encryption and the private key for decryption. The method to generate the keys comes from the difficulty of factoring the product of two large prime numbers. Due to the computational complexity of the algorithm, RSA is fairly slow for encryption or decryption, and thus it is not often applied to encrypt the data or message directly. It is mostly used to encrypt the secret key for symmetric key cryptography and to send it to the recipient. The recipient will decrypt to retrieve the secret key and use that secret key to decrypt the encrypted message.

RSA algorithm contains four steps: encryption, decryption, key generation and key distribution [20, 286-287].

- Key generation: The goal is to create the public key and private key based on two large prime numbers.

  Below here is the general concept of computing public key and private key.

  1. Choose randomly and independently two distinct large prime number **p** and **q**.
  2. Compute n = pq, which n is used as the modulus of both public key and private key.
  3. Compute the Euler's totient function $\varphi(n) = (p-1)(q-1)$
  4. Choose an integer $1<e<\varphi(n)$ which is co-prime to $\varphi(n)$

5. Compute d such as de mod φ (n) = 1
   The public key contains the modulus n and the public exponent e
   The private key contains the modulus n and the private exponent d

- Key distribution: To enable the sender to send the encrypted messages, the recipient shares the public key to the sender via some reliable channels. The private key must not be shared.
  The public key (n, e) is dc shared to the sender.
- Encryption: The sender encrypts the message with the public key.
  Before encryption, the message is encoded as a number (called m).
  The cipher text is calculated: $c = m^e \pmod n$. The value c is sent to the recipient.

- Decryption: The recipient decrypts the message with the private key.
  The plaintext is calculated: $m = c^d \pmod n$. The value m is decoded into the message.

3.2.2 Digital Signature

Digital signature is a cryptographic data string which is related to a message or document in digital form and used to verify the authenticity of that message or document. **Digital signature scheme** is the cryptographic mechanism to generate and verify the digital signature. It contains key generation algorithm, digital signature generation algorithm (signing algorithm) and digital signature verification algorithm. [20, 425.]

In reality, the digital signature concept is applied widely in many fields, especially in information security. The certification of public keys is one of the most important applications of the digital signature on the internet. With the certification, an entity or a party can prove its own identity to the other parties. Moreover, the digital signature also encounters the principles of cryptography, which are authentication, integrity and non-repudiation.

Besides the encryption purpose, RSA algorithm also has the mechanism to generate the digital signature from the message and verify it from the others. The key generation algorithm was described in section 3.2.1. The digital signature generation algorithm and digital signature verification algorithms inherit the public key and private key values to perform the tasks [20, 433-435].

- Signature generation: The signature is computed by the cryptographic hash function between the message and the private key. The digest value is the signature and included in the message when sent.
  The signature called s: $s = m^d \pmod n$.

- Signature verification: The receiver uses the same hash function with the public key and the signature to compute the substring of the message. It compares the digest with the signature and rejects the message if it is not equal.
  The digest s': $s' = s^e \pmod n$
  Compare s' and s.

4 Digital Certificate

4.1 Overview

Digital certificate, also known as a public key certificate, is an electronic credential used to verify the ownership of a public key. It also includes the information of the owner's identity and the digital signature to ensure the contents are correct. If the certificate and the signature are trustworthy, the public key can be used to communicate with the owner of the digital certificate [24]. Figure 8 illustrates the information of a SSL certificate viewing from Firefox browser.

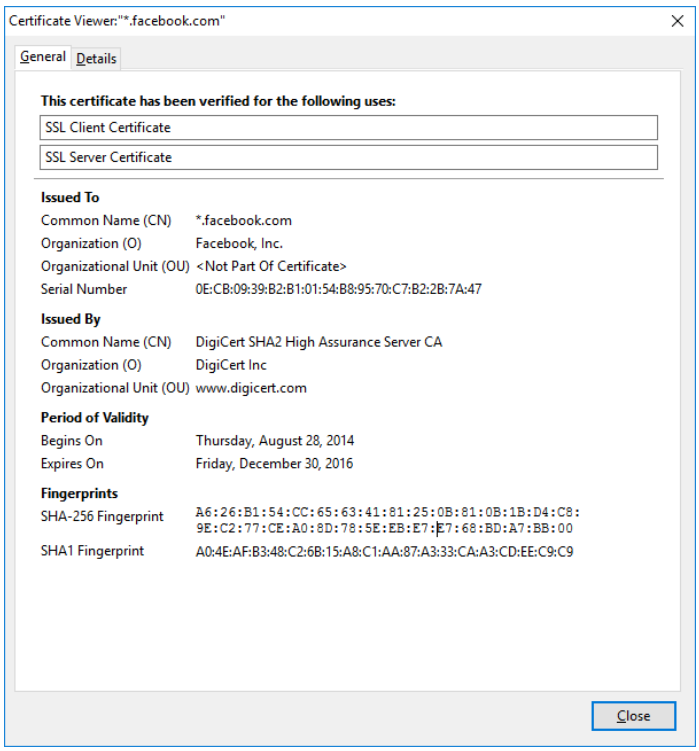

Figure 8. Facebook.com SSL certificate viewing from Firefox browser

A **public key Infrastructure** (PKI) is a set of roles, procedures and policies to manage the digital certificates. The main purpose of the PKI is to distribute the public keys securely over the insecure channels. The certificate may be revoked if it is discovered that its related private key has been compromised, or if the relationship between an entity and a public key embedded in the certificate is discovered to be incorrect or has changed. [24.] A typical PKI contains:

- A certificate authority (CA) issues the digital certificates.
- A registration authority (RA) verifies the identity of the entities that request their digital certificates.
- A certificate management system.
- One or more directories store the certificates.

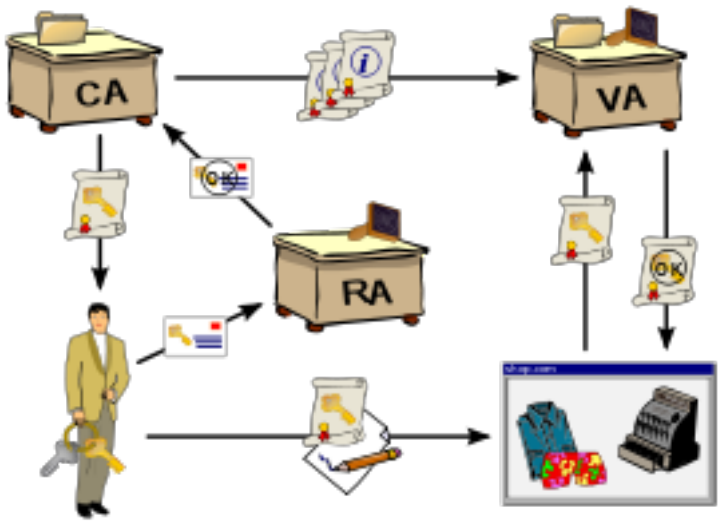

Figure 9. Diagram of a public key infrastructure. Copied from Chrkl. [25]

A certificate authority (CA) is capable of digitally signing the certificate from the public key and its associated entity. The digital certificate is signed by the CA's own private key; therefore it is trusted and validated by the CA [24].

4.2 X.509 Standard

X.509 is the standard for the public key infrastructure issued in 1988. It is initially associated with the X.500 standard, developed by ITU-T. This standard uses a hierarchical system of certificate authorities (CAs) for issuing the certificate. The 3rd version of X.509 currently supports the other topologies such as bridges and meshes [26, 10; 26, 13]. X.509 indicates the formats of the digital certificates, certificate revocation lists (CRL) attribute certificates and the certification path validation algorithm [27, 11].

According to the RFC 2459 document, the digital certificate structure of the X.509 system is as follows [27]:

- Certificate
  - Version: v1, v2 or v3
  - Serial Number
  - Signature
  - Issuer Name
  - Validity
    - Not Before

        - Not After
    - Subject Name
    - Subject Public Key Info:
        - Algorithm Identifier
        - Subject public key
    - Issuer Unique ID (optional)
    - Subject Unique ID (optional)
    - Extensions (optional)
- Signature Algorithm
- Signature Value

The common filename extensions for X.509 certificates are the following [28]:

- .pem – (Privacy-enhanced Electronic Mail) Base64 encoded DER certificate, enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----"
- .cer, .crt, .der – usually in binary DER form, but Base64-encoded certificates are common too (see .pem above)
- .p7b, .p7c – PKCS#7 SignedData structure without data, just certificate(s) or CRL(s)
- .p12 – PKCS#12, may contain certificate(s) (public) and private keys (password protected)
- .pfx – PFX, predecessor of PKCS#12 (usually contains data in PKCS#12 format, e.g., with PFX files generated in IIS).

A certificate revocation list (CRL) is a list of certificates which have been revoked by their CA; thus the entities presenting those certificates are no longer to be trusted. According to the RFC 5280 document, the reasons for revoking a certificate are listed as below [29, 69-69]:

- Unspecified (0)
- Key Compromise (1)
- CA Compromise (2)
- Affiliation Changed (3)
- Superseded (4)
- Cessation of Operation (5)
- Certificate Hold (6)
- Remove from CRL (8)

- Privilege Withdrawn (9)
- AA Compromise (10).

The CRLs often bring a digital signature associated with their CA and published periodically.

4.3 Client Certificate Authentication

4.3.1 Overview

A **Client certificate** is used to identify a client or a user. Fundamentally, it represents a user identity and provides the user with authentication to the server. Unlike SSL server certificates, the client certificate is not used for encrypting or decrypting the data at all. Client certificate is issued to a user by a certificate authority and it contains both public key and private key portions. The certificate authority could be a widely known organization that offers the certificate services or it could be an internal or private server that is only used by the company.

Client certificate authentication, also known as Client SSL authentication, is the feature that lets one authenticate the users who are accessing the server by exchanging the client certificate. It excludes anonymous and unauthorized access to the server side. The client authentication occurs when the server requests the client certificate during the SSL handshake. The client certificate sent to the server will be verified by the CA which can be both self-signed or external one. [30, 55]

When the client certificate authentication is set up from the server side, it will require the client certificate sent from the client side. If the certificate is not provided or invalid, the server will decline the request and the user will receive a corresponding error message.

Generally, both sides have a list of root certificate authorities (Root CA) they trust. When the server side prompts for a client certificate, the trusted Root CA list is included in the request. The client side also has its own trusted Root CA list and it will compare its list with the server side list to create a list of matched Root CAs. After that, the client selects the suitable client certificate issued from the list of matched Root CAs and sends it back to the server. The server validates the client certificate and decides if the communication between the client and server can continue.

Table 1. Comparison between client and server authentication.

	Client SSL authentication	**Server SSL authentication with basic password authentication**
Require the user to enter a name and password	No	Yes
Encrypts transactions over the network, identities the server, and validates message sent	Yes	Yes
Validates user identity	Yes	Not validated by a 3rd party
Portable to other computers	No-browser software contains the certificate	Yes – user can enter the name and password on any machine
Name and password encrypted over the network to prevent eavesdropping	n/a	Yes

Table 1 shows the differences between the client authentication and server authentication. The only disadvantage of the client authentication is the difficulty when porting and installing the certificate to other computers. Ordinary users might have troubles to port or install the client certificate and authenticate themselves to the service. However, the client SSL certificate authentication is the perfect choice for mobile applications that can be programmed to install the certificates automatically. The mobile application authenticates and identifies itself to the server before accessing it. Third parties like the inauthentic mobile application or the web browser is not able to authenticate and access it. The client SSL certificate authentication creates a solid security layer to the mobile application.

4.3.2 Self-signed Certificate Creation with OpenSSL

OpenSSL is an open source project that provides a robust, and full-featured toolkit for Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It contains the software library for general-purpose cryptography and is written in the C programming language. The OpenSSL license is Apache License, therefore the OpenSSL toolkit is free to use for commercial and non-commercial purposes [31].

In order to perform the client certificate authentication, the server needs to own a Root CA certificate and the client needs to send the Client certificate which is signed by the Root CA certificate. Both the certificates can be created by using OpenSSL commands.

For Windows OS environment, the OpenSSL toolkit can be run from the Command Prompt with Administrator right [31].

The process of creating the root CA certificate is the standard process for creating any self-signed certificate.

Step 1: Create the Root CA Certificate

- Generate the private key for the Root CA Certificate as figure 10 shows.

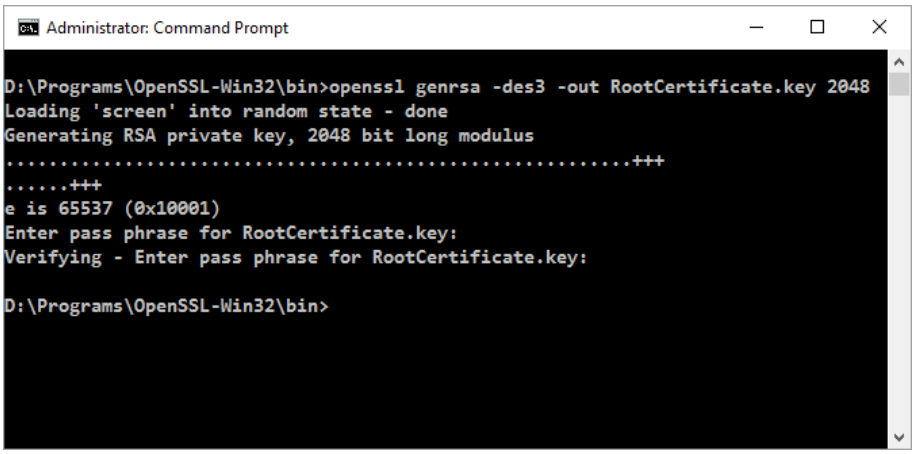

Figure 10. Root CA Certificate private key.

genrsa is the function to generate an RSA private key

-des3 is the symmetric-key block cipher

2048 is the size of private key in bits

- Create the public key from the private key and enter the certificate information as figure 11 shows.

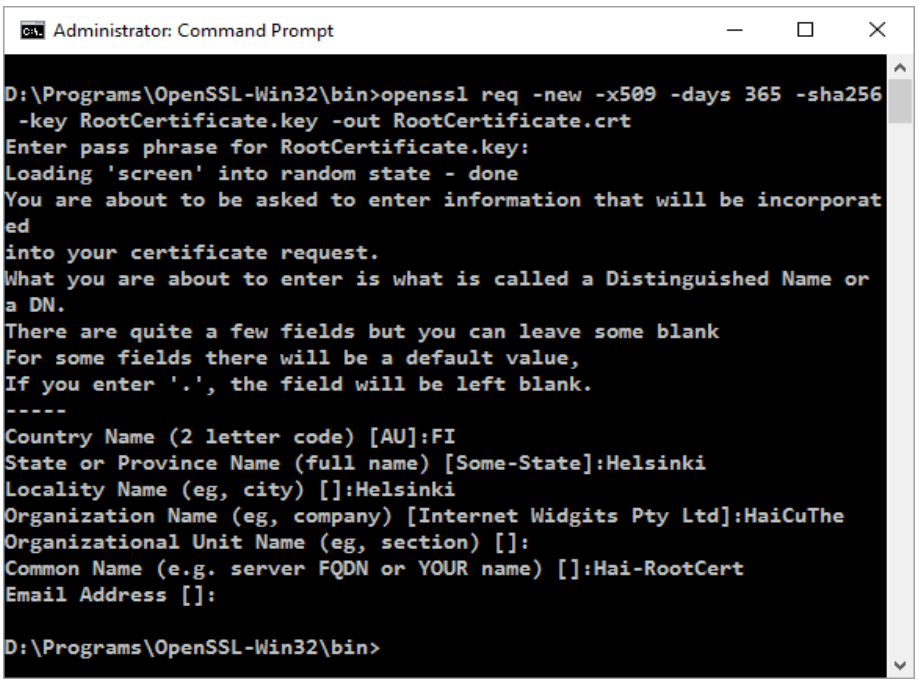

Figure 11. Root CA Certificate public key.

req is the function to generate PKCS#10 certificate request and certificate.

-new is the option to generate a new certificate request

-x509 is the option to generate the self signed certificate instead.

-days 365 specifies the number of days to certify the certificate for (365 days in this example)

-sha256 is the cryptographic hash function

- Create the Personal Information Exchange (PFX) file by combining the private key and public key as figure 12 shows.

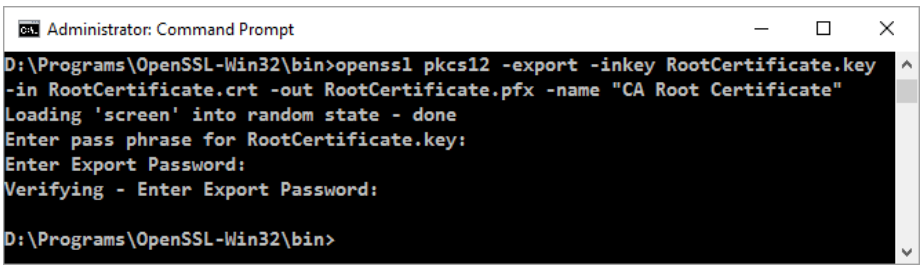

Figure 12. Root CA Certificate PFX.

pkcs12 is the function for PKCS#12 file

Step 2: Create the certificate signing request (CSR) for the Client certificate

- Generate the private key for the Client certificate as figure 13 shows.

Figure 13. Client Certificate private key.

- Generate the CSR file for the Client certificate based on the private key. Figure 14 illustrates the declared information of the to-be-signed certificate.

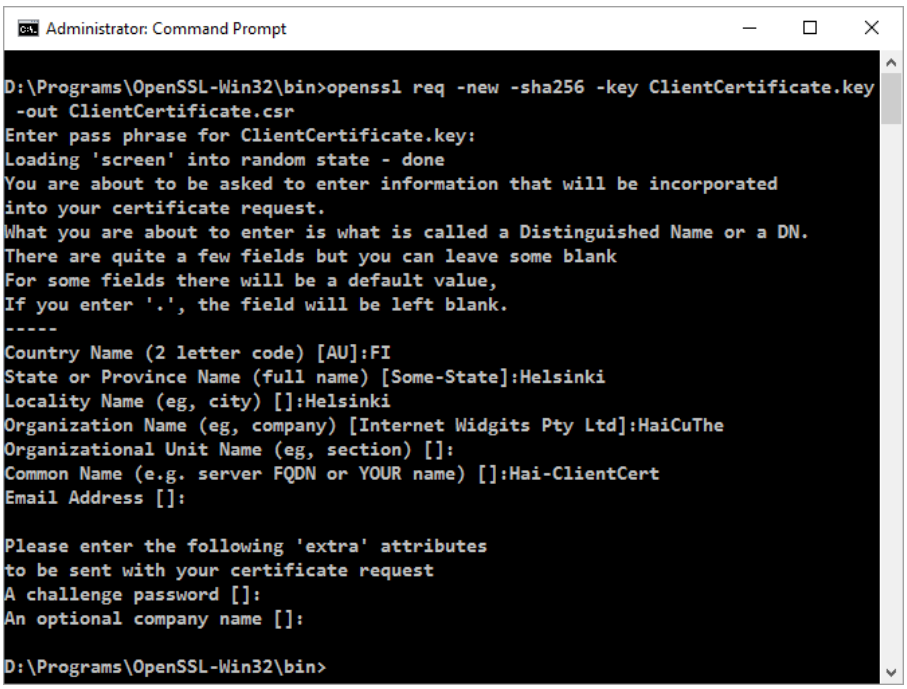

Figure 14. Client Certificate CSR.

without **–x509** parameter, the **req** function turns from self-signed certificate generation to CSR generation

Step 3: Create the Client certificate from the CSR file.

- Use the Root CA Certificate private and public key to sign the client certificate as in figure 15.

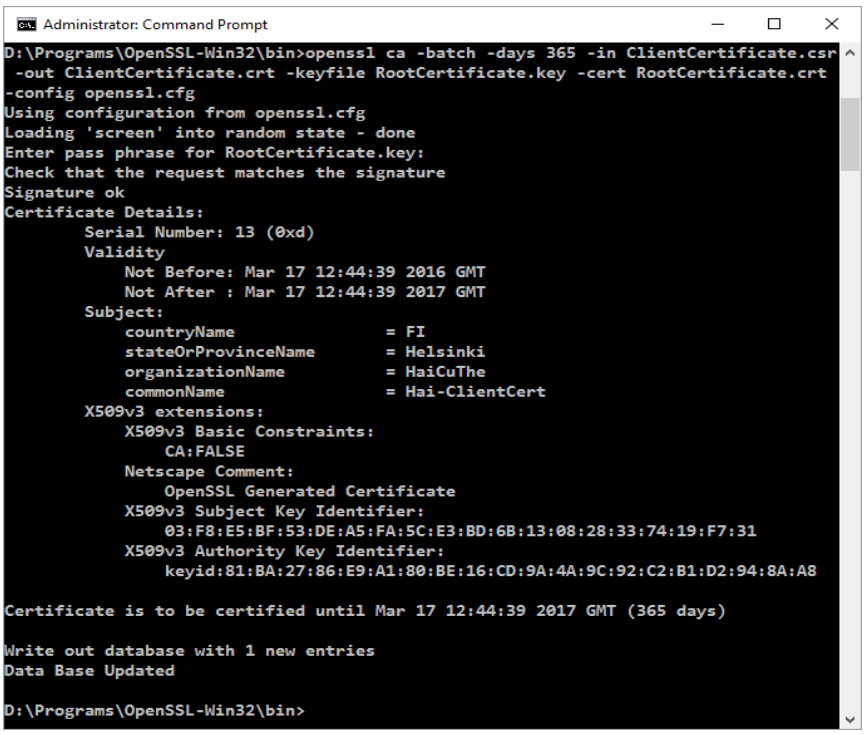

Figure 15. Client Certificate signing.

ca is the function to sign certificate requests

-config specifies the configuration file to use

- Create the Personal Information Exchange (PFX) file by combining the private key and public key.

After the process is done, it is possible to recognize the difference between the client certificate and the Root CA certificate from the general information.

Figures 16 - 19 show the client certificate and the root CA certificate are both issued by “Hai-RootCert”. The root certificate is self-signed; therefore the **Issued by** value is itself. Meanwhile the client certificate is signed by the root certificate, thus the **Issued by** value is the root CA certificate.

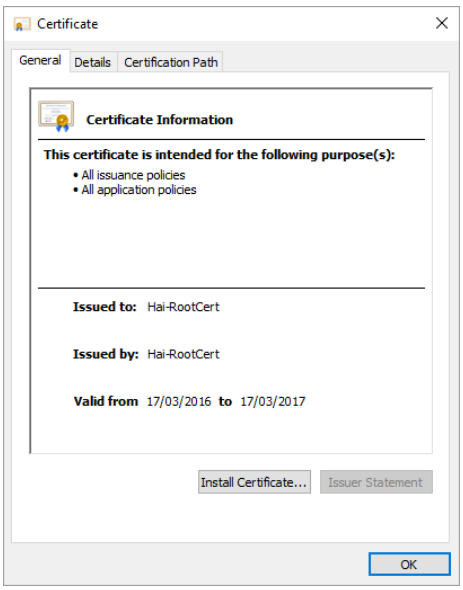

Figure 16. Root Certificate General Info

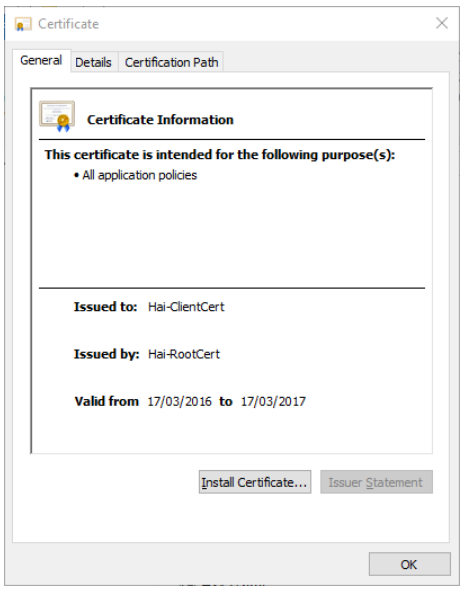

Figure 17. Client Certificate General Info

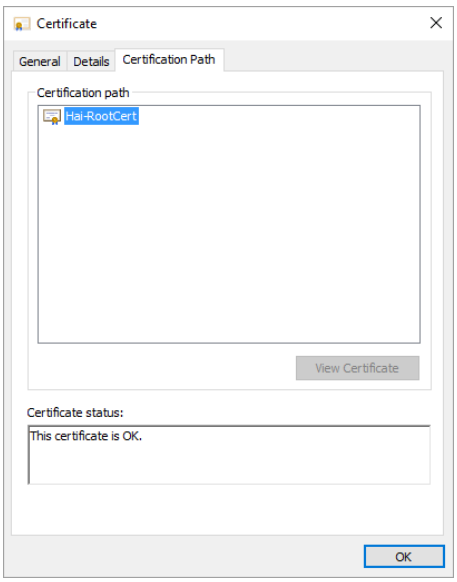

Figure 18. Root Certificate Path

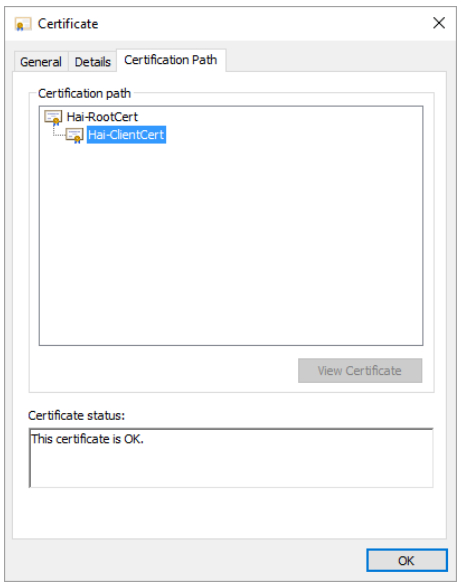

Figure 19. Client Certificate Path

4.4 Server Certificate

4.4.1 Overview

A **Server certificate**, also known as **SSL certificate** or **Server SSL certificate**, is used to identify a server. Usually, it represents the server identity such as a machine name or a host header (for example: www.haicuthe.com). Server certificate is used to encrypt

and decrypt the content between the client side and server side. Typically, the production server certificate is issued by the well-known organization that offers the certificate authority only [30, 46-49]. The self-signed certificate is not accepted by the client side like the web browser and mobile phone, and the server or web service using that server certificate is marked as untrusted. In the server certificate, the **Issued to** value specifies the hostname which it has been issued.

4.4.2 Hypertext Transfer Protocol and Transport Layer Security

Secure Socket Layer (SSL) and its successor **Transport Layer Security** (TLS) are the cryptographic protocols, designed for securing the communication over the computer network. The purpose of TLS protocol is to provide privacy and data integrity between two communicating applications [30, 5]. The TLS protocol consists of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol offers the connection security with two basic properties [30, 15: 30, 26]:

- The connection is private because the symmetric cryptography is applied for data encryption.
- The connection is reliable because the message integrity check is included in the message transport (computed by secure hash functions).

The TLS Handshake Protocol offers connection security with three properties:

- The peer's identity can be authenticated using the public key cryptography. It requires at least one of the peers' authentications to perform the handshake.
- The negotiation of a shared secret is secure.
- The negotiation is reliable.

HTTPS, also known as **HTTP Secure** or **HTTP over TLS**, is a well-known protocol for securing communication over computer network. It contains the application-level Hypertext Transfer Protocol (HTTP) for collaborative, distributed, hypermedia information system within the connection encrypted by Transport Layer Security (TLS) [32, 2]. It creates a secure channel to protect the data from eavesdroppers and man-in-the-middle attacks, offers the cipher suites and ensures that the server certificate is verified and trusted. The public and private keys of the server certificate are used to generate the shared secret which is applied to encrypt the data between the client and server.

4.4.3 Process to Obtain a Server Certificate

The server certificate is used for HTTPS. Hence, it is necessarily signed by the CA certificate of well-known organizations. Table 2 shows the usage percentage and the market share of the CAs

Table 2. Usage percentage and market share of the CAs. Copied from W3techs.com [33]

Certificate Authorities	**Usage percentage**	**Market share**
Codomo	7.8%	40.8%
Symantec	5.1%	26.9%
GoDaddy	2.3%	12.1%
GlobalSign	1.8%	9.7%
DigiCert	0.6%	3.0%
StartCom	0.4%	2.1%
IdenTrust	0.4%	2.0%
Verizon	0.1%	0.5%

In the self-signed process (section 4.3.2), the certificate signing request (CSR) is signed with the self-signed CA created. However, for the server certificate creation, the CSR of the server certificate is submitted to the genuine CA instead.

Step 1: Create the CSR file for the host header as in figure 20.

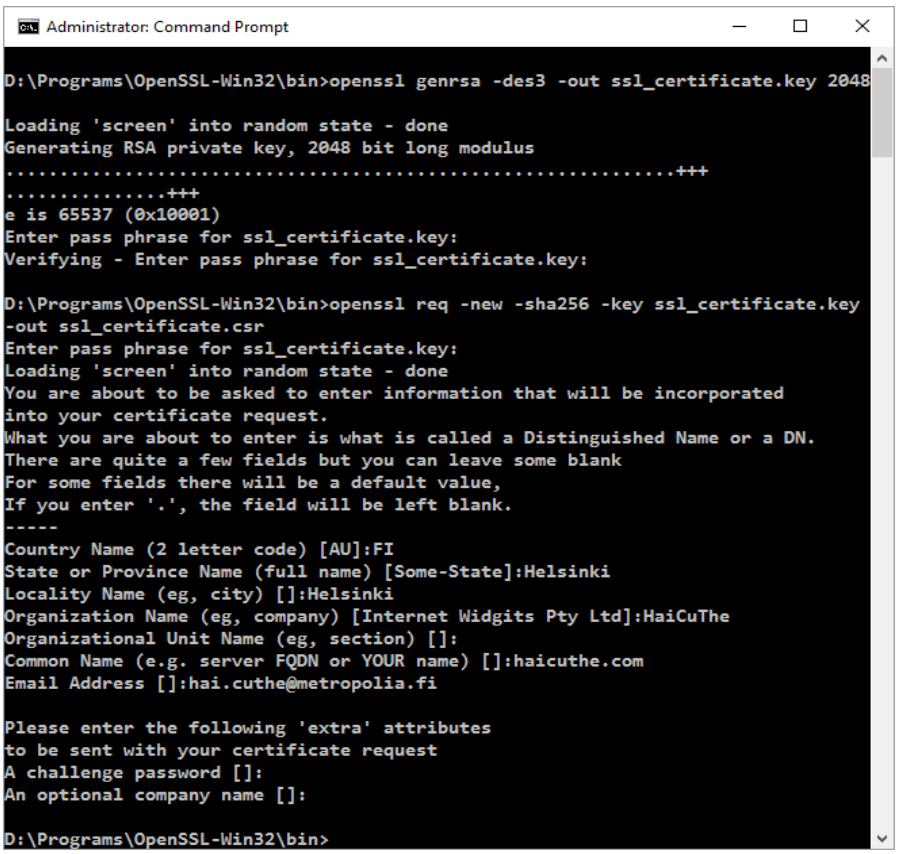

Figure 20. Creating CSR for SSL certificate

Step 2: Submit the CSR to the CA

In figure 21, the Symantec CA is selected for CSR submission.

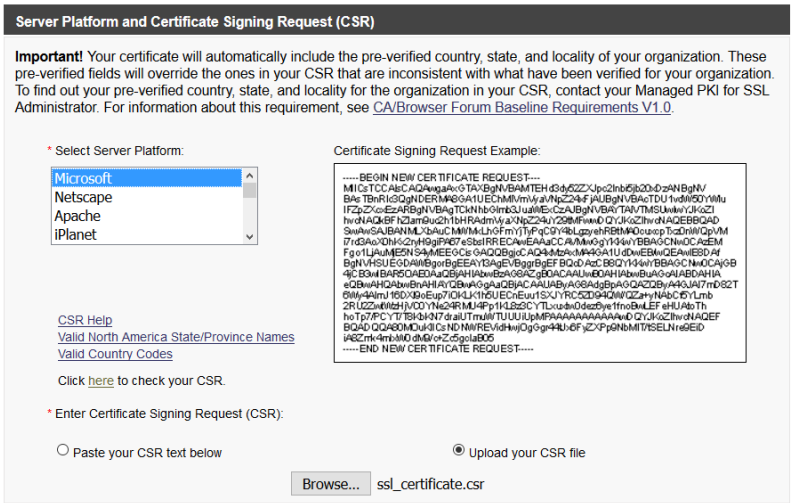

Figure 21.CSR submission on Symantec site.

The server is run on Azure Cloud Service; therefore, the server platform is Microsoft. The CSR file is also uploaded.

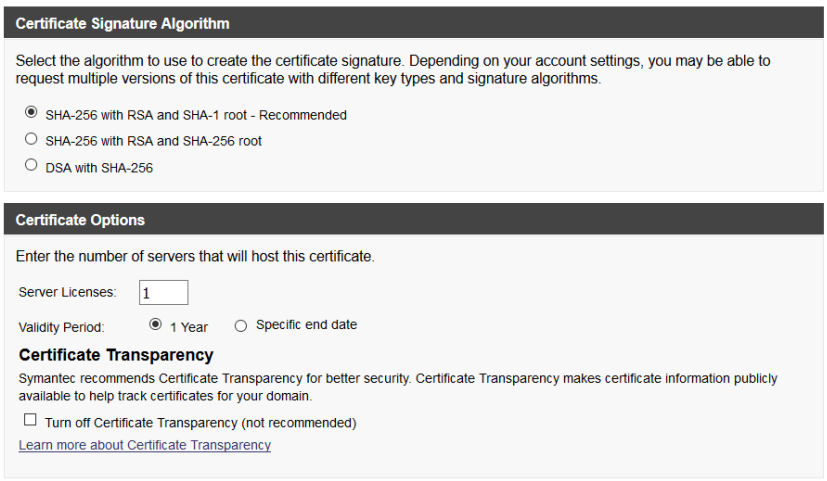

Figure 22. CSR submission options

As figure 22 shows, there are a few options for certificate signing such as certificate signature algorithm and validity period. After selecting the options, press submit to send the submission to the CA.

Step 3: Verify the ownership of the domain name

After submitting the signing request, Symantec Validation team will contact the owner of the domain (in this case is haicuthe.com) to verify the ownership of the domain and the certificate request. Once verified, the SSL certificate is issued and sent to the owner via email. It can also be downloaded from the Symantec site in the owner account. Finally, it is ready to be installed on the server.

5 Case Study

5.1 Introduction

In this thesis, a proof-of-concept server-client Windows Phone application is implemented to demonstrate the capability of exchanging messages securely without being breached or facing other security attacks. Although the application runs on the Windows Phone platform and the server is hosted and runs in Windows Azure, the general idea of

creating secure communication channel and encrypting messages between the two parties is concrete. It can be applied to many different mobile platforms such Android or iOS applications.

The main feature of the proof-of-concept application is to send confidential information or message to the server. The server stores the information in the database and responses to the application if the information is accepted or rejected. The application utilizes the crypto theories and security concepts mentioned in the section 3 and 4 to encrypt the message in the application before sending it to the server.

In this case study, the terms “the client”, “the client side” or “the application” have the same meaning and refer to the mobile application that wants to send the requests to the server.

5.2 HTTPS and Client Certificate Authentication

In the proof-of-concept application, the server is configured to use the HTTPS protocol with a certificate signed by the authentic CA. HTTPS protects the communication channel from eavesdropping, tampering and man-in-the-middle attacks. It guarantees which the server that the client applications want to communicate with is the correct one and ensures the contents of the communication between them without being interfered by any means.

The client certificate authentication is set up as the barrier from the unauthenticated clients. The main purpose is to identify if the requests to the server come from the original application. Only the original application with the client certificate installed internally is able to connect to the server. Any attempts connecting to the server without the client certificate are considered as the unauthorized access from the attackers; therefore they are dropped immediately and the IP addresses of the access origin are also blocked temporarily. The client certificate authentication is not usually a popular choice for authentication in a web application because of the difficulty of the certificate installation. However, in a native mobile application, the client certificate can be installed the first time the application is launched. Therefore, the client certificate authentication is one of viable choices. The drawback of this solution is that the application cannot generate the client certificate itself. Therefore, the client certificate can only be the application’s identity, not the actual user’s identity.

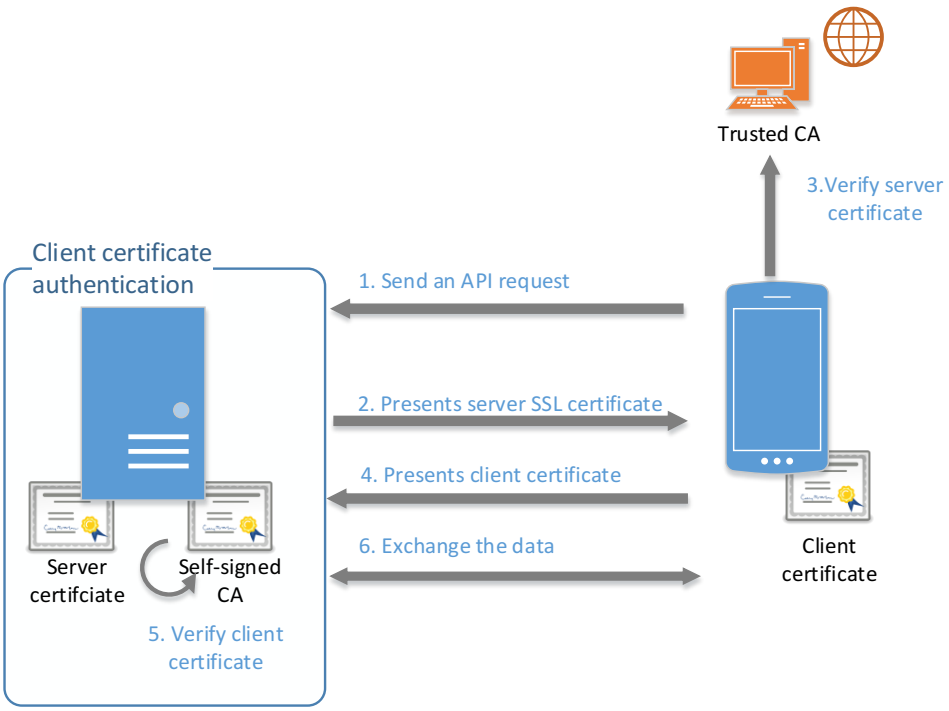

Figure 23. Client - Server authentication process

Figure 23 illustrates the authentication process between the application and the server. The prerequisites for the process are:

- The Server SSL certificate is installed on the server. Its common name must be the domain name of the server.
- The self-signed CA is also installed on the server.
- The client certificate is signed by the self-signed CA above and installed in the application.

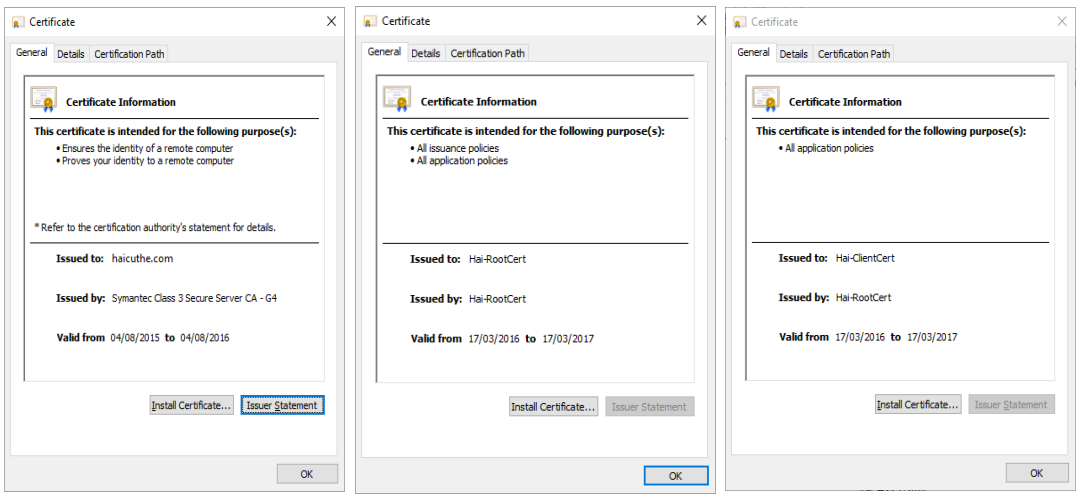

Figure 24. Server SSL certificate, self-signed CA, client certificate

The steps of the authentication process are specified below:

1. The client sends the API request to the server.
2. The server responses with its own SSL certificate to prove its identity.
3. In the client, the SSL certificate is verified by comparing the certificate's CA with the client's trusted CA list. In the Windows Phone platform, it is the Internet Explorer's trusted CA list.
4. If the client trusts the server with the SSL certificate, the client will present its own client certificate to authenticate.
5. The server uses the self-signed CA to verify the client certificate.
6. When the verification is a success, the communication channel between the client and server will be established. They can start to exchange data and messages securely.

5.3 Encrypting and Decrypting Messages

The encryption and decryption system of the application is inspired by the OpenPGP standard for encrypting and decrypting data. The system uses a combination of symmetric key encryption and public key encryption to form the encrypted message. The system is built on the decentralized trust mode "web of trust" concept, which does not rely on a certificate authority but the independent users with their own identity certificates. [34.]

The system consists of two parts: encryption and decryption. Encryption is the process in the application side, while decryption is in the server side. To be able to send an encrypted message to the server, the application needs to have the server's public key, thus the public-private key pair is generated beforehand. The server owns the private key to decrypt and the application uses the public key to encrypt. Figure 25 demonstrates the flow of encryption and decryption in the system.

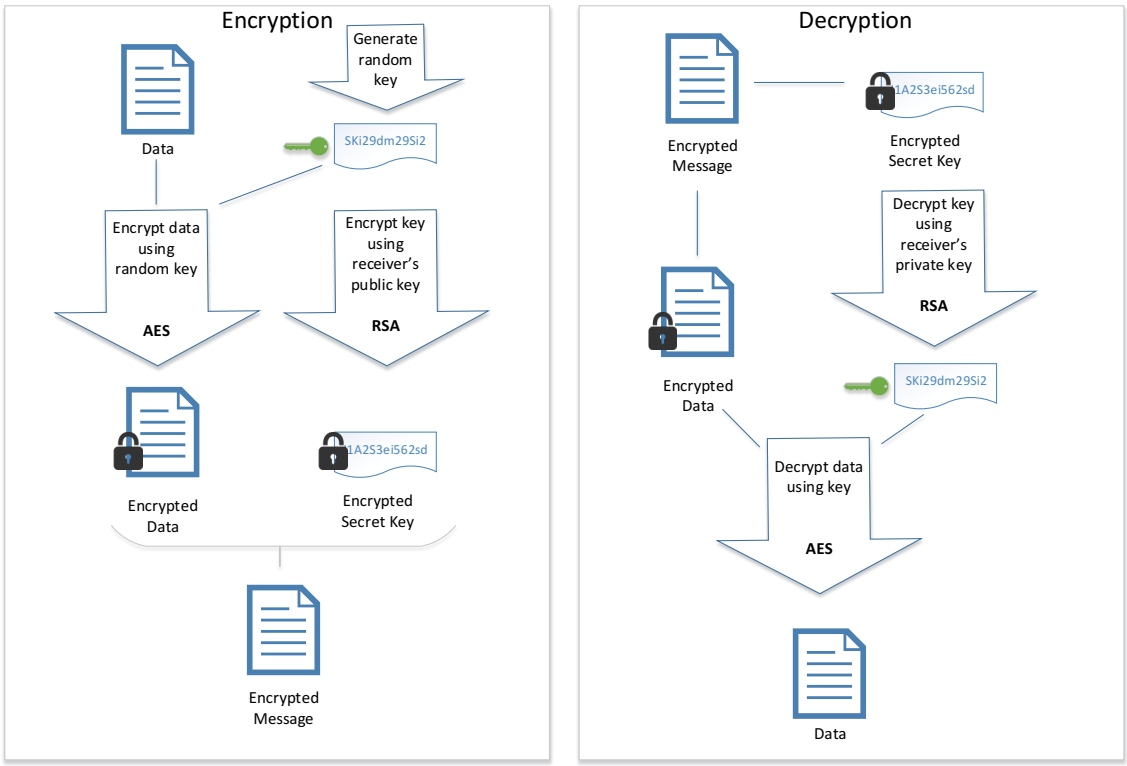

Figure 25. Encryption and decryption system

The steps for the application encryption are specified below:

1. The user writes a plaintext message.
2. A random secret key is generated from the application side.
3. Encrypting the plaintext message with the random secret key using symmetric key encryption.
4. The random key itself is also encrypted by the server's public key using public key encryption.
5. Serializing them into the encrypted message and be ready to send to the server.

In the server, the steps are described below:

1. De-serializing them into the encrypted data and the encrypted key.

2. Decrypting the secret key with the server's private key.
3. Using the secret key to decrypt the encrypted data into the original message.
4. The server stores the message in the database.

In Windows Phone 8.1 application development, the PFX certificate file is not considered the source file. Therefore, it is not included into the application package when it is complied. The workaround for this problem is changing the file extension of the certificate file from PFX to TXT to fool the application compiler. The new TXT certificate file can still be installed as usual and also complied into the binary code which cannot be decompiled.

5.3.1 Symmetric Key Encryption

The secret key is generated randomly in the client side.

```
public static string CreateSymmetricSecretnKey()
{
      // Open a symmetric algorithm provider for the AES algorithm with
CBC mode.
      SymmetricKeyAlgorithmProvider objAlg =
SymmetricKeyAlgorithmProvider.OpenAlgorithm(SymmetricAlgorithmNames.Ae
sCbc);

      // Create a symmetric secret key.
      IBuffer keyMaterial = CryptographicBuffer.GenerateRandom(32);
      CryptographicKey secretKey =
objAlg.CreateSymmetricKey(keyMaterial);

      var buffSecretKey = keyMaterial;
      return CryptographicBuffer.EncodeToBase64String(buffSecretKey);
}
```

Listing 1. Symmetric Secret Key function

Listing 1 shows a secret key is generated for the AES algorithm with the CBC mode of operations. The result is encoded into the Base 64 string for friendly user read.

```
public static string AESEncrypt(string dataToEncrypt, string
secretKey, string password)
{
  //Create AES algorithm with 256 bit key and 128-bit block size
  AesManaged aes = new AesManaged();
  ProtectedMemory.Unprotect(secretKey,
MemoryProtectionScope.SameLogon);
  //Add salt to sercetKey
  Rfc2898DeriveBytes rfc2898 = new Rfc2898DeriveBytes(password,
secretKey);
  ProtectedMemory.Protect(secretKey,
MemoryProtectionScope.SameLogon);

  aes.Key = rfc2898.GetBytes(aes.KeySize / 8);
  ProtectedMemory.Protect(aes.Key, MemoryProtectionScope.SameLogon);
  rfc2898.Reset(); //needed for WinRT compatibility
  aes.GenerateIV(); //generate IV
  aes.Mode = CipherMode.CBC;

  //Create Memory and Crypto Streams
  MemoryStream memoryStream = new MemoryStream();
  CryptoStream cryptoStream = new CryptoStream(memoryStream,
aes.CreateEncryptor(),CryptoStreamMode.Write);

  //Encrypt Data
  byte[] data = Encoding.UTF8.GetBytes(dataToEncrypt);
  cryptoStream.Write(data, 0, data.Length);
  cryptoStream.FlushFinalBlock();

  //Return encrypted data in string
  string encryptedStr = ByteToHexString(memoryStream.ToArray());
  return encryptedStr;
}
```

Listing 2. AES Encryption function

Listing 2 describes the symmetric key encryption function, which the secret key needs to go through the Rfc2898DeriveBytes constructor to add a salt and password before using it as the key for encrypting and decrypting. The AES algorithm also generates the dynamic IV and uses the CBC mode of operation.

```
public static string AESDecrypt(string dataToDecrypt, string
secretKey, string password)
{

      //Create AES algorithm with 256 bit key and 128-bit block size
      AesManaged aes = new AesManaged();
      ProtectedMemory.Unprotect(secretKey,
      MemoryProtectionScope.SameLogon);
      Rfc2898DeriveBytes rfc2898 = new Rfc2898DeriveBytes(password,
      secretKey);
      ProtectedMemory.Protect(secretKey,
      MemoryProtectionScope.SameLogon);

      //Decrypt Data
      byte[] dataToDecryptByte = HexStringToByte(dataToDecrypt);

      aes.Key = rfc2898.GetBytes(aes.KeySize / 8);
      ProtectedMemory.Protect(aes.Key,
      MemoryProtectionScope.SameLogon);
      rfc2898.Reset(); //needed for WinRT compatibility
      byte[] IV = new byte[aes.IV.Length];
      Array.Copy(dataToDecryptByte, 0, IV, 0, IV.Length);
      aes.IV = IV;
      aes.Mode = CipherMode.CBC;

      //Create Memory and Crypto Streams
      MemoryStream memoryStream = new MemoryStream();
      CryptoStream cryptoStream = new CryptoStream(memoryStream,
      aes.CreateDecryptor(), CryptoStreamMode.Write);

      cryptoStream.Write(dataToDecryptByte, aes.IV.Length,
      dataToDecryptByte.Length);
      cryptoStream.FlushFinalBlock();

      //Return Decrypted String
      byte[] decryptBytes = memoryStream.ToArray();
      decryptedText = Encoding.UTF8.GetString(decryptBytes, 0,
      decryptBytes.Length);
}
```

Listing 3. AES Decryption function

On the contrary, the symmetric key decryption function decrypts the encrypted message in a reverse order as listing 3 shows.

5.3.2 Public Key Encryption

The prerequisite of public key encryption is having the public-private key pair for encrypting and decrypting. The public key is distributed to the application beforehand and the private key is kept in the server.

```
public static string RSAEncrypted(X509Certificate2 x509_2, string
PlainStringToEncrypt)
{
      string PlainString = PlainStringToEncrypt.Trim();
      byte[] cipherbytes = ASCIIEncoding.ASCII.GetBytes(PlainString);
      // Using Public key for encryption
      RSACryptoServiceProvider rsa =
      (RSACryptoServiceProvider)x509_2.PublicKey.Key;
      byte[] cipher = rsa.Encrypt(cipherbytes, false);
      return Convert.ToBase64String(cipher);
}
```

Listing 4. RSA Encryption function

```
public static string RSADecrypted(X509Certificate2 x509_2, string
EncryptedString)
{
      byte[] cipherbytes =
      Convert.FromBase64String(EncryptedStringToDecrypt);
      if (cipherbytes.Length % 16 != 0)
      {
          throw new DataDecryptException("Bad length of the data");
      }
      if (x509_2.HasPrivateKey)
      {
          // Decrypt with the private key
          RSACryptoServiceProvider rsa =
      (RSACryptoServiceProvider)x509_2.PrivateKey;
          byte[] plainbytes = rsa.Decrypt(cipherbytes, false);
          System.Text.ASCIIEncoding enc = new
      System.Text.ASCIIEncoding();
          return enc.GetString(plainbytes);
      }
      else
      {
          throw new Exception("Certificate used for has no private
      key.");
      }
}
```

Listing 5. RSA Decryption function

Listing 4 and 5 demonstrates the encryption and decryption functions in the server and client side.

5.4 Usage

5.4.1 Server Side

After deploying to the Azure cloud service, the server is up and ready to be accessed from the application. The domain name (haicuthe.com) needs to be redirected to the server's IP address. The server SSL certificate should also be installed in the server side.

- Accessing the server via a web browser without the client certificate would be denied.

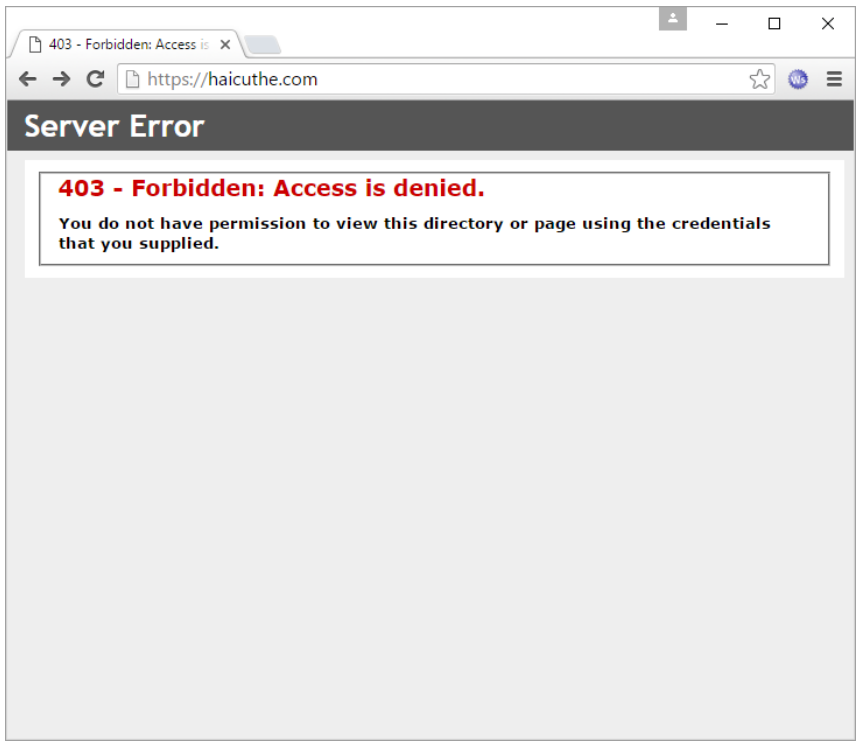

Figure 26. Screenshot of denied access in Google Chrome.

- Accessing the server with the client certificate would be successful.

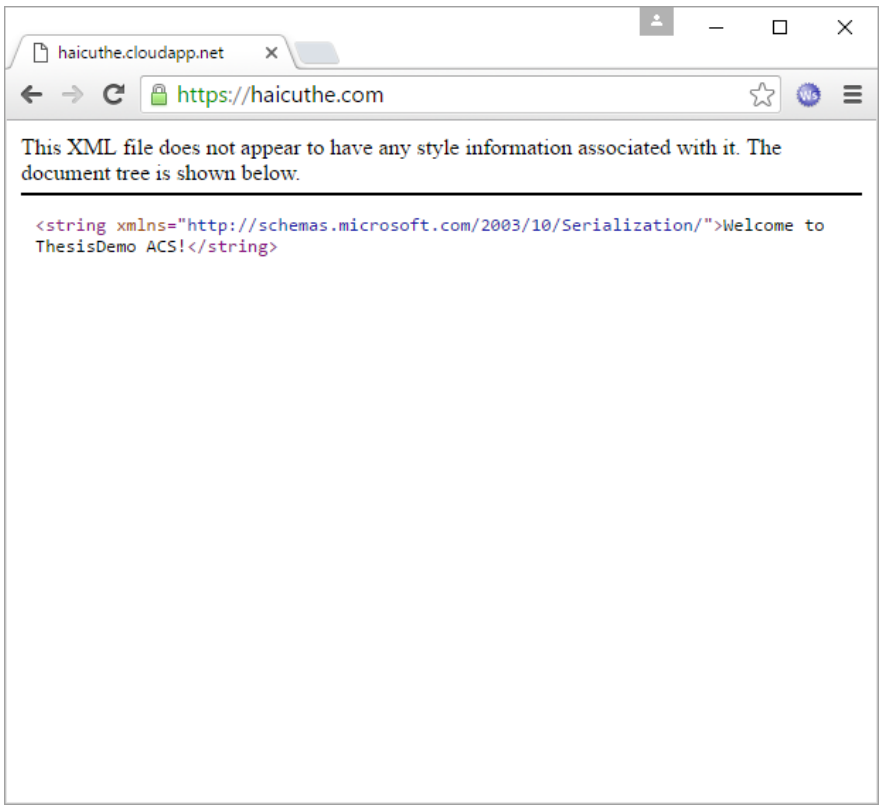

Figure 27. Screenshot of successful access in Google Chrome.

The server has two APIs:

- GET /home/index: the default API when accessing the server. It returns the string "Welcome to ThesisDemo ACS!".

- POST /home/send: this application sends the request to this API and the server responses back with the decrypted message and decrypted secret ket.

5.4.2 Client Side

A proof-of-concept Windows Phone application is implemented to demonstrate message encryption. Figure 28 is a screenshot of the Windows Phone application.
It contains two textboxes, two call-to-action (CTA) buttons and two text blocks.

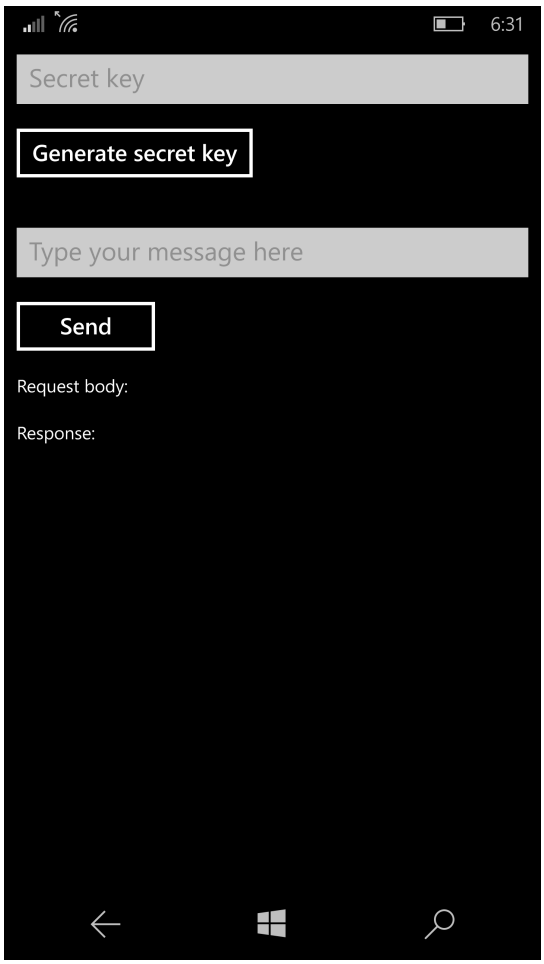

Figure 28. The proof-of-concept Windows Phone application

Figure 28 shows the components listed as below:

- Secret key textbox: It can be entered manually or generated automatically from the CTA button. The length of the secret key needs to be exactly 256 bits; therefore, it is recommended to generate it automatically.

- Message textbox: The user enters his/her message that he/she wants to send to the server. The length of the message can be varied.

- The “Send” CTA button is to trigger sending the “POST /home/send” API request to the server.

- The two last lines are to display the encrypted message that the application sends to the server and the decrypted message from the server’s response.

Figure 29. Example result of the application.

Figure 29 shows the request body and response body in the JSON format:

- The request body contains the encrypted data and encrypted secret key.
- The response body contains the decrypted data and decrypted secret key.

The mechanism and the source code of message encryption and decryption were explained in section 3.1 and 3.2.

Figure 30. Another sample result of the application

The length of the original message and the encrypted message is proportional, thus, it once again proves that the symmetric key encryption is needed to encrypt the message.

5.5 Security Analysis

There is no perfectly secure system. Therefore, it is necessary to prepare a system from attackers. Table 3 describes the possible scenarios and how to mitigate the problems.

Table 3. Scenario list

Scenario	**Explanation**	**Mitigation**
Decompile app package	The Windows Phone 8.1 app package is decompiled	The client certificate is not exposed because it's hidden under the TXT file extension.

	by some professional software. It exposes the source code of the app.	
The client certificate is compromised.	The attacker hacks the app and gets the certificate.	The certificate can be revoked from the server side. All users would not access the server anymore until the app update was published to the App store.
Eavesdropping and man-in-the-middle attacks	The traffic between the app and Azure Cloud Server is intercepted and the token is extracted.	The traffic is encrypted with HTTPS.
The certificate authority is compromised	The CA private key is exposed	The message between the client and server is still not exposed and decrypted as long as the recipient's private key is kept secret
The secret key is exposed		The hacker cannot replicate the encrypted message without the public-private key pair.
The recipient's private key is exposed	The developer's PC might be hacked and lost the private key into the hacker	The server needs to be stopped. Update the server and the application with new public-private key pair.

Table 3 only lists the foreseeable scenarios. There might be more vulnerabilities and loopholes in the system; therefore, the system should be tested thorough from the third party.

6 Conclusion

The main purpose of this thesis was to study cryptography systematically in the application security field applying the theories to implement a secure web service for a mobile application. It started from the basic concepts to the advanced ones and explained the best practice algorithms in a simple way. The case study was a proof-of-concept Windows Phone application with the web service hosted in the Azure cloud. All the mentioned algorithms and best practices were applied and implemented to create a secure web service with multiple layers of security.

In the case study, a mobile application was programmed in the Windows Phone platform and the server was hosted in the Azure cloud. However, the security solution could be applied to other mobile platforms such as iOS or Android. The cryptographic libraries

related to encryption and decryption in Objective C and in Java were implemented differently from .NET but the usage and the result remains the same. The only potential problem for Android and iOS platform was installing the client certificate and using it for authentication from the third party application point of view.

Finally, the study showed the importance of security and cryptography in the information technology field and especially in software development. Security is a process and not a product. The current best practice might be broken in the next few years. Therefore, developers need to keep up with the latest knowledge to improve the security of their products.

References

1 Microsoft News. Microsoft Unveils Windows Phone 7 Series [online]. Redmond, Washington, U.S.; February 2010.
URL: https://news.microsoft.com/2010/02/15/microsoft-unveils-windows-phone-7-series/. Accessed 9 April 2016.

2 Microsoft News. Nokia and Microsoft Announce Plans for a Broad Strategic Partnership to Build a New Global Mobile Ecosystem [online]. Redmond, Washington, U.S.; February 2011.
URL: https://news.microsoft.com/2011/02/10/nokia-and-microsoft-announce-plans-for-a-broad-strategic-partnership-to-build-a-new-global-mobile-ecosystem/. Accessed 9 April 2016.

3 IDC. Smartphone OS Market Share, 2015 Q2 [online]. Framingham, Massachusetts, U.S.; July 2015.
URL: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Accessed 9 April 2016.

4 Microsoft News. Microsoft Announces Broad Availability of Handheld PCs with Windows CE [online]. Redmond, Washington, U.S.; November 1996.
URL: http://news.microsoft.com/1996/11/19/microsoft-announces-broad-availability-of-handheld-pcs-with-windows-ce/. Accessed 9 April 2016.

5 Microsoft. Windows Embedded Compact 2013 [online]. Redmond, Washington, U.S.; September 2014.
URL: https://msdn.microsoft.com/en-us/library/gg154201.aspx. Accessed 9 April 2016.

6 Microsoft News. Microsoft Rings in Pocket PC Phone Edition [online]. Redmond, Washington, U.S.; February 2002.
URL: http://news.microsoft.com/2002/02/19/microsoft-rings-in-pocket-pc-phone-edition/. Accessed 9 April 2016.

7 Pollze A. Windows CE - A Contrasting Approach & Embedded Systems with Windows XP Embedded [online]. Potsdam, Germany: Hasso-Plattner Institute; June 2015.
URL: https://www.tele-task.de/archive/podcast/21360/. Accessed 9 April 2016.

8 Microsoft News. Microsoft Unveils Windows Mobile 2003 Software for Pocket PCs [online]. Redmond, Washington, U.S.; June 2003.
URL: http://news.microsoft.com/2003/06/23/microsoft-unveils-windows-mobile-2003-software-for-pocket-pcs/. Accessed 9 April 2016.

9 Microsoft News. Microsoft Releases Windows Mobile 5.0 [online]. Redmond, Washington, U.S.; May 2005.
URL: http://news.microsoft.com/2005/05/10/microsoft-releases-windows-mobile-5-0/. Accessed 9 April 2016.

10 Epstein Z. Apple and Google dominate smartphone space while others scramble [online]. BGR, U.S.; December 2011.
URL: http://bgr.com/2011/12/13/apple-and-google-dominate-smartphone-space-while-other-vendors-scramble/. Accessed 9 April 2016.

11 Microsoft News. Microsoft Unveils Windows Phone 7 Series [online]. Redmond, Washington, U.S.; February 2010.
URL: https://news.microsoft.com/2010/02/15/microsoft-unveils-windows-phone-7-series/. Accessed 9 April 2016.

12 Juozaityte L. AdDuplex Windows Phone Device Statistics Report for February, 2016 [online]. Vilnius, Lithuania: Adduplex. February 2016.
URL: http://www.slideshare.net/adduplex/adduplex-windows-phone-statistics-report-january-2016. Accessed 9 April 2016.

13 Microsoft News, Microsoft showcases latest updates to Windows, opportunities for developers [online]. Redmond, Washington, U.S.; April 2014.
URL: https://news.microsoft.com/2014/04/02/microsoft-showcases-latest-updates-to-windows-opportunities-for-developers/. Accessed 9 April 2016.

14 Microsoft, Minimum hardware requirements [online]. Redmond, Washington, U.S.;
URL: https://msdn.microsoft.com/en-us/library/windows/hardware/dn915086(v=vs.85).aspx. Accessed 9 April 2016.

15 Microsoft News. Windows 10: The next chapter [online]. Redmond, Washington, U.S.;
URL: https://news.microsoft.com/windows10story/. Accessed 9 April 2016.

16 Microsoft. Get started with Windows 10 Mobile [online]. Redmond, Washington, U.S.;
URL: http://windows.microsoft.com/en-us/windows-10/getstarted-whats-new-cortana-mobile. Accessed 9 April 2016.

17 Microsoft. Windows Bridge for iOS [online]. Redmond, Washington, U.S.;
URL: https://dev.windows.com/en-us/bridges/ios. Accessed 9 April 2016.

18 Microsoft. Windows Phone architecture overview [online]. Redmond, Washington, U.S.; July 2015.
URL: https://sysdev.microsoft.com/en-us/Hardware/oem/docs/Getting_Started/Windows_Phone_architecture_overview. Accessed 9 April 2016.

19 Microsoft. Security for Windows Phone 8 [online], Redmond, Washington, U.S.;
URL: https://msdn.microsoft.com/en-us/library/windows/apps/ff402533(v=vs.105).aspx. Accessed 9 April 2016.

20 Menezes A, Oorschot P, Vanstone S. Handbook of Applied Cryptography. Florida, U.S.: CPC Press; 1997.

21 FIPS PUB 197. Specification for the Advanced Encryption Standard [online]. Gaithersburg, MD, U.S.: The National Institute of Standards and Technology; November 2001.
URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. Accessed 9 April 2016.

22 FIPS PUB 197. Specification for the Advanced Encryption Standard [online]. Gaithersburg, MD, U.S.: The National Institute of Standards and Technology; November 2001.

URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. Accessed 9 April 2016.

23 Ferguson N, Schneier B, Kohno T. Cryptography Engineering. Indianapolis, IN, U.S.: Wiley Publishing; 2010.

24 Tycksen F, Jennings C. Digital Certificate – Patent US6189097 B1 [online]. Sunnyvale, CA, U.S.: Preview System Inc. March 1997.
URL: https://www.google.com/patents/US6189097. Accessed 9 April 2016.

25 Chris. Principle of a public key infrastructure [online]. August 2007.
URL: https://commons.wikimedia.org/wiki/File:Public-Key-Infrastructure.svg. Accessed 9 April 2016.

26 Cooper M, Dzambasow Y, Hesse P, Joseph S, Nocholas R. Internet X.509 Public Key Infrastructure: Certification Path Building [online]. IETF. September 2005.
URL: https://tools.ietf.org/html/rfc4158. Accessed 9 April 2016.

27 Housley R, Ford W, Polk W, Solo D. Internet X.509 Public Key Infrastructure Certificate and CRL Profile [online]. IETF. January 1999.
URL: https://tools.ietf.org/html/rfc2459. Accessed 9 April 2016.

28 Turner S. Asymmetric Key Packages [online]. IETF. August 2010.
URL: https://tools.ietf.org/html/rfc5958. Accessed 9 April 2016.

29 Cooper D, Santesson S, Farrell S, Boeyen S, Housley R, Polk W. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [online]. IETF. May 2009.
URL: https://tools.ietf.org/html/rfc5280. Accessed 9 April 2016.

30 Dierks T, Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.2 [online]. IETF. August 2008.
URL: https://tools.ietf.org/html/rfc5246. Accessed 9 April 2016.

31 OpenSSL. OpenSSL Cryptography and SSL/TLS Toolkit User Guide [online]. February 2016.
URL: http://openssl.org/docs/fips/UserGuide-2.0.pdf. Accessed 9 April 2016.

32 Rescorla E. HTTP Over TLS [online]. IETF. May 2000.
URL: https://tools.ietf.org/html/rfc2818. Accessed 9 April 2016.

33 W3Techs. Usage of SSL certificate authorities for websites [online].
URL: http://w3techs.com/technologies/overview/ssl_certificate/all. Accessed 9 April 2016.

34 Callas J, Donnerhacke L, Finney H, Shaw D, Thayer R. OpenPGP Message Format [online]. IETF. November 2007.
URL: https://tools.ietf.org/html/rfc4880. Accessed 9 April 2016.

Appendix 1: Source Code

The main source code of proof-of-concept Windows Phone application.

Listing 6 includes the XAML source code for MainPage screen (MainPage.xaml)

```
<Page
    x:Class="ThesisDemoApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:ThesisDemoApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <StackPanel>
        <TextBox x:Name="tbMsg" PlaceholderText="Type your message
here" Margin="10"></TextBox>
        <StackPanel Orientation="Horizontal">
            <TextBox x:Name="tbKey" PlaceholderText="Secret key"
Margin="10"></TextBox>
            <Button x:Name="btnGenerateKey" Content="Generate key"
HorizontalAlignment="Center" Click="btnGenerateKey_Click" ></Button>
        </StackPanel>
        <Button x:Name="btnSend" Content="Send"
HorizontalAlignment="Center" Click="btnSend_Click"></Button>

        <TextBlock x:Name="tbResponse" Text="Response:"></TextBlock>
    </StackPanel>
</Page>
```

Listing 6. MainPage.xaml source code

Listing 7 includes the C# code behind MainPage screen.

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
using ThesisDemoApp.Encryptors;
using ThesisDemoApp.Models;
using ThesisDemoApp.WebServices;

namespace ThesisDemoApp
{

    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();

            this.NavigationCacheMode = NavigationCacheMode.Required;
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
        }



        private async void btnSend_Click(object sender,
RoutedEventArgs e)
        {
            MessageRequest msgRequest = new MessageRequest();
            string result = await
ThesisDemoService.Service.Send(msgRequest);
            tbResponse.Text += result ?? "failed!";
        }

        private void btnGenerateKey_Click(object sender,
RoutedEventArgs e)
        {
            var sessionKey =
PublicKeyEncryption.CreateSymmetricKeyPair();
            tbKey.Text = sessionKey;
        }
    }
}
```

Listing 7. MainPage.xaml.cs source code

Listing 8 shows the ClientCertificateHelper.cs source code for installing digital certificates

```
public class ClientCertificateHelper
    {
        public static string Thumbprint =
"DB69055521078B012EDD708E013E926837188D22";

        public static async void InstallCertificate()
        {
            string certPassword = string.Empty;
            string certName = string.Empty;

            System.Uri clientCertificateFile = new System.Uri("ms-
appx:///Assets/Certs/ClientCertificate.txt");
            Windows.Storage.StorageFile clientFile = await
Windows.Storage.StorageFile.GetFileFromApplicationUriAsync(clientCerti
ficateFile);
            Windows.Storage.Streams.IBuffer buffer = await
Windows.Storage.FileIO.ReadBufferAsync(clientFile);
            string clientCertificateData =
Windows.Security.Cryptography.CryptographicBuffer.EncodeToBase64String
(buffer);

            // now install the certificate in the certificate store of
the app
            await
CertificateEnrollmentManager.ImportPfxDataAsync(clientCertificateData,
                certPassword, // password for ClientCertificate.txt
                ExportOption.NotExportable, // there is no reason to
keep the certificate Exportable
                KeyProtectionLevel.NoConsent, // whether any consent
is required
                InstallOptions.None, // no installation options
                certName); // use the same friendly name as in the
certificate request for future reference
        }

    }
```

Listing 8. ClientCertificateHelper.cs source code